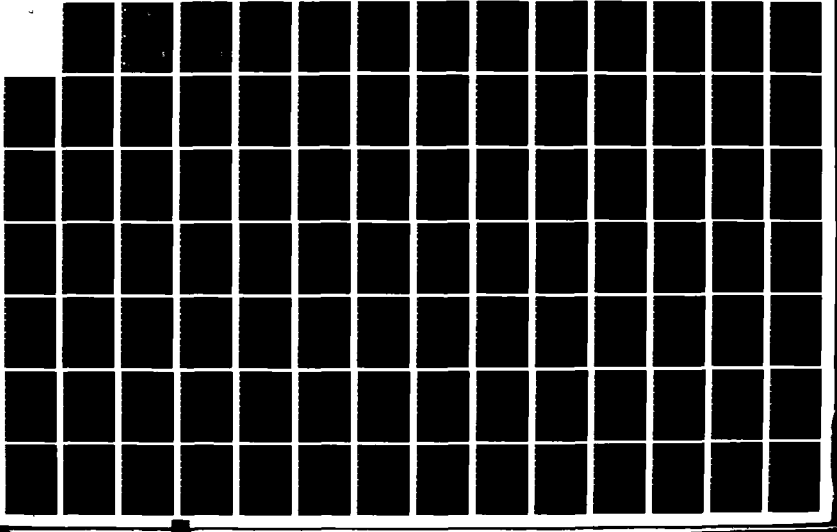
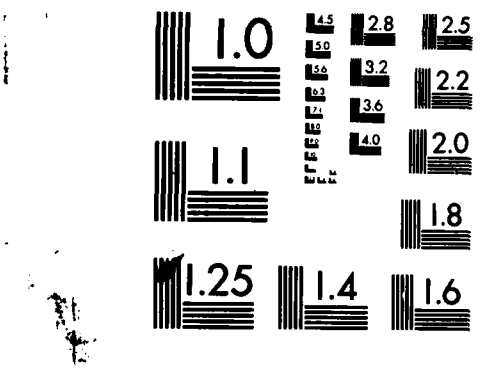
AD-A138 025 INTERACTIVE COMPUTER GRAPHICS FOR SYSTEM ANALYSIS(U) 1/3
AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOOL
OF ENGINEERING M A TRAVIS DEC 83 AFIT/GE/EE/83D-66
UNCLASSIFIED F/G 5/1 NL





INTERACTIVE COMPUTER GRAPHICS FOR SYSTEM ANALYSIS

THESIS

AFIT/GE/EE/83D-66 Mark A. Travis
CAPT USAF

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY
AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT/GE/EE/83D-66

INTERACTIVE COMPUTER GRAPHICS

FOR SYSTEM ANALYSIS

THESIS

AFIT/GE/EE/83D-66 Mark A. Travis
CAPT USAF

Approved for public release; distribution unlimited

AFIT/GE/EE/83D-66

INTERACTIVE COMPUTER GRAPHICS FOR SYSTEM ANALYSIS

THESIS

Presented to the Faculty of the School of Engineering
of the Air Force Institute of Technology
Air University
in Partial Fulfillment of the
Requirements for the Degree of
Master of Science

by
Mark A. Travis
CAPT USAF
Graduate Electrical Engineering
December 1983

ACKNOWLEDGEMENTS

The possibilities of computer graphics in system analysis are almost unlimited. Modern graphics can provide a wealth of information to the user that is not only easy to interpret but even pleasant to use. A good graphics program for system analysis would be a priceless tool for both students and professional designers. It is easy to imagine how much quicker I would have learned system analysis had such a tool been available. These personal and professional interests of myself and the AFIT faculty have fostered this effort.

I would like to thank several individuals who were most helpful during this effort. Many thanks go to Dr. Gary Lamont for guidance and direction in this work while still permitting me to make the important decisions in design and implementation. I also wish to thank Dr. Robert Fontana and Professor Charles Richard for offering their expertise in control systems and graphics. Thanks also to Mr. John Smith (ASD) for comments and assistance and to Mr. Bob Ewing and Mr. Dick Wager for keeping the VAX up and running.

Finally, I must thank my wonderful wife, Candy. Her cherished assistance and support throughout this effort and all of the AFIT tour has made life much more pleasant.

TABLE OF CONTENTS

TABLE OF CONTENTS (cont'd)

ABSTRACT

The possible applications of computer graphics for control system design are considered in this thesis. The functional requirements of such a system are identified and discussed. Other topics such as graphics software, programming languages, user interface, and software design are vital to such an effort and are examined in detail. Based on the desirable features of device independent graphics, an implementation of the ACM/SIGGRAPH Core Standard Proposal is selected. To demonstrate the various applications for system analysis, a group of functions for analysis of continuous systems is implemented and tested. Sample results are provided and recommendations for further work are discussed.

INTERACTIVE COMPUTER GRAPHICS FOR SYSTEM ANALYSIS

CHAPTER I

INTRODUCTION

The purpose of this effort is to define, develop, and evaluate an interactive computer graphics tool for control system design and analysis. This investigation will draw heavily on the results of many individual efforts in interactive computing, computer analysis of control systems, and computer graphics.

System analysis is a technique frequently used by design engineers and students to determine the response or output of a system to a particular input signal. It is most often employed in the design of communication or control systems. The classical analysis method normally specifies the system transfer function in the complex frequency domain and involves analysis of the root locus and how it affects the system's response. Inverse Laplace or Fourier transforms are performed to obtain responses in the time domain. This classical method is still very important in both the classroom and design lab and it is nearly always

used in teaching the basic concepts of system analysis. However, system design using this method usually involves a large number of repetitive and time consuming calculations. In some cases, the design is an iterative task; i.e. the transfer function is slightly changed, by "moving" its poles and zeros, and the new response is analyzed and compared to a desired response. If the results are not satisfactory, the process is repeated. These repetitive and monotonous tasks are perfectly suited to the modern computer.

Until the modern digital computer was fully developed and readily available to a large number of users, system analysis and design was accomplished using pencil, paper, and lots of "elbow grease." Many designers performed analog simulations in conjunction with the pencil and paper work [1:402]. The analog simulations had the added benefit of providing output in a graphical manner via oscilloscopes, spectrum analyzers, and strip chart recorders. This graphical data added new meaning to the designers' equations and hand calculations. However, the iterative design process was still very time consuming because each change of the transfer function would often require mechanically changing connections or component values.

The digital computer and modern programming languages can easily accomplish the repetitive calculations, but the general purpose computer does not provide graphical output to the user. The traditional output from a digital computer

is a list of data points presented in a columnar format. However, with the appropriate software and hardware, computer graphics can be used to display the results in a manner that contains more information and is easier to interpret than a list of data points [29:2]. Using computer graphics for system analysis is the main thrust of this investigation.

BACKGROUND

The Air Force Institute of Technology (AFIT) School of Engineering, under the direction of Dr. Gary Lamont, has supported several efforts concerned with development of an integrated computer-aided design tool for control systems. Stanley J. Larimer [17] created a comprehensive design tool known as TOTAL which is currently hosted on the CDC Cyber at Wright Patterson Air Force Base. It is used most often by students studying control systems design, the Flight Dynamics Lab, and the Aeronautical Systems Division. However, since TOTAL performs a variety of operations (for example, polynomial factoring and matrix inversions) it is frequently used by students in other disciplines.

Successive efforts by Glen Logan [19] and Charles Gembarowski [9] resulted in the development of ICECAP (Interactive Control Engineering Computer Analysis Package). Logan's work was primarily concerned with implementing TOTAL

on a Digital Equipment Corporation VAX 11/780. Gembarowski followed this effort by adding to and modifying the VAX TOTAL to improve the user interface. His work centered on defining and developing a command language for TOTAL. The result was ICECAP which is currently undergoing more development. The use of a command language represents a significant improvement over the standard version of TOTAL which is driven by option numbers input by the user. This type of menu selection for TOTAL is somewhat cumbersome as the current number of options is in excess of 140 [18:A84-A86]. Gembarowski also worked on defining and implementing an on-line help function with hopes of eventually eliminating a user's manual.

There are currently two other related efforts under the supervision of Dr. Lamont. Robert Wilson is expanding the original ICECAP implemented by Gembarowski and Vincent Parisi is examining the possibilities for implementing ICECAP or a similar system on a micro-computer.

The Air Force Institute of Technology, also under the direction of Dr. Lamont, has sponsored several individual efforts in the area of computer graphics. The purpose of the work was to implement a graphics package on a VAX 11/780 so AFIT students could have a facility on which to study graphics concepts and perform graphics research. Philip Tarbell [37] reviewed several packages, most of which were developed around the proposed graphics standard prepared by

the Special Interest Group on Graphics of the Association for Computing Machinery [36]. This standard is known as the Core System, and although it was never adopted by the American National Standards Institute, it has been implemented by several colleges, laboratories, and manufacturers. Tarbell concluded that a FORTRAN implementation of the Core System by George Washington University was best suited for AFIT's VAX 11/780 since it was developed on a VAX and all available documentation pertained to the VAX [37:118]. This particular implementation is referred to as the GWCORE and their February 1981 version, which is currently in use on the AFIT VAX, does not contain the input functions as set forth in the ACM proposed standard [37:120]. Another AFIT project, though, conducted by Harold Curling in 1980 resulted in the design of these graphics input functions. In 1982, another AFIT student, Kevin Rose, continued the study of the ACM Core standard, graphics input, and display interfaces. He developed a graphics editor known as INGRED which permits a user to create, edit, and save two dimensional drawings. In addition, he developed an interface (known in graphics terms as a device driver) for a color raster display. His work demonstrated one of the most important concepts of the Core standard which is its ability to generate device-independent pictures [29:78].

PROBLEM STATEMENT

Although TOTAL and ICECAP are very powerful tools for control system analysis and synthesis, interactive computer graphics can be used to enhance the information provided from the computer to the user. The primary reason for using computer graphics in any application is the speed in which the user can interpret and understand the displayed data. With the ability to interact with a computer, the system designer can easily correct or modify his design and quickly observe the response of his new design [25:5-6]. The primary objective of this effort is to define, develop, and evaluate a compuer graphics tool for performing system analysis.

TOTAL and ICECAP do have some forms of graphics output but the user interface is somewhat less than ideal. Both permit viewing of a root locus and time response on a video display but the graphics are created using standard alphanumeric characters and only one can be displayed at any time. If the user wishes to change a system parameter, the current display is lost, making it difficult to compare results. Although it is possible to obtain a hard copy of the results using a line printer or high resolution plotter, it may be several hours before the output is available in a batch processing environment. TOTAL also has the capability to generate high resolution video displays using the

Tektronix 4010 family of terminals. The PLOT10 graphics package [28], a Tektronix commercial product, is used by TOTAL to create the plots [19:231]. Since PLOT10 is designed for the Tektronix terminal, this graphic function of TOTAL is not device-independent and therefore may not be able to be used at other installations not using the Tektronix display.

Gembarowski [9] has already considered several aspects of the user interface for TOTAL. The same concepts must apply to graphics programs. A good user interface makes the program both easy to learn and easy to use [25:13]. This is especially true for a student who is still learning the concepts of control theory and system analysis. The computer aided design tool, particularly for this user, must truly be an aid rather than an entirely new discipline. This effort aims to define and develop a tool which minimizes or overcomes these various problems.

SCOPE

Although discrete time systems are not significantly more complex, the computational methods are somewhat different so this investigation is concerned primarily with continuous time systems. However, if time permits, implementation of matrix operations and discrete time system analysis will be considered. The basic concepts of interactive graphics, user interface, and software

engineering would apply equally to a design tool for discrete time systems.

Although device independent graphics is a primary consideration of this work, the only high resolution display available in AFIT's Digital Engineering Lab is a Tektronix 4014. Thus, output to other displays will not be considered and new device drivers will not be developed. Output to hard copy devices such as plotters or line printers will be accomplished only if suitable device drivers exist.

The classical approach to system analysis is the only design method considered in this work. For this method, the design tool will permit analysis of the root locus, frequency response, and time response.

APPROACH

A logical three phase approach is used for this study: requirements definition, implementation, and evaluation.

The requirements definition phase consists primarily of a review of TOTAL and ICECAP to determine the basic requirements of a computer aided design package for control systems. In addition, the necessary requirements for interactive graphics, software engineering and human factors will be defined.

The implementation phase of the study transforms the

requirements into an actual design. Proven software engineering techniques will be employed in developing the required functions. The development will be in a modular style and completely documented with a user's manual and data dictionary to permit additions or maintenance to the package.

The evaluation phase is divided into two parts. The first part is a functional test to insure that the package performs as desired. All display formats, system messages, and mathematical functions will be tested. The results of the mathematical tests will be compared to hand calculations and other computer results such as those from TOTAL and ICECAP. All problems or deficiencies will be documented. The second part of the test phase will evaluate the user interface. Although these types of tests are often very subjective, techniques are being devised to measure this vague quality [31:125-126].

OVERVIEW OF THESIS

The next chapter of this thesis contains the conceptual requirements for a basic system analysis tool. Chapter Three presents the details of the complete design and how this design was developed based on the requirements. The fourth chapter documents the implementation of the design. Chapter Five discusses the test strategy and the sixth and final chapter includes a summary, conclusion, and

recommendation. The appendices of this thesis contain a programmer's manual, a brief user's manual, and sample test results. Volume II, the source code listings for this project, is maintained in the Digital Engineering Laboratory.

CHAPTER II

REQUIREMENTS DEFINITION

INTRODUCTION

This chapter develops the conceptual requirements of a computer graphics program for system analysis. The requirements are divided into four major categories. The first category is system analysis. All of the functional requirements fall under this category. The remaining three categories; computer graphics, software engineering , and human interface, are considered general design requirements. While these categories do not affect the functional requirements, they do play a major role in creating an abstract mental picture of the system before the actual design begins. Each category is presented in detail and, at the end of the chapter, the complete functional requirements are summarized.

SYSTEM ANALYSIS

Past efforts [9, 19, and 17] identified a complete set of functional requirements for a computer aided control system design tool. These requirements encompass conventional, modern, and stochastic design techniques for continuous and discrete systems. As an example, conventional analysis of continuous time systems is

discussed in the following paragraphs:

Typically, a computer graphics program for system analysis should permit the user to perform conventional design and analysis of the control system shown in Figure II-1.

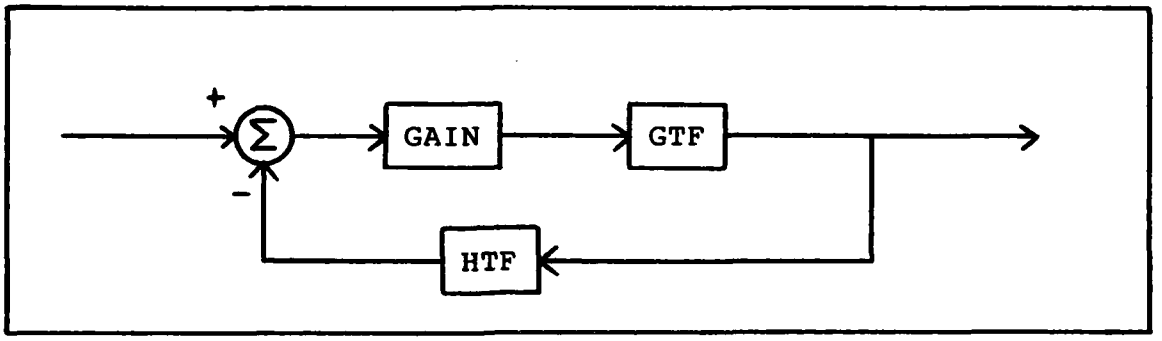


Figure II-1. Typical Feedback Control System

Most of the characteristics of this feedback control system can be determined by analyzing the open loop and closed loop transfer functions. Note that the nomenclature and the labeling of transfer functions is similar to that used by TOTAL [18]. This is done wherever possible so that users already familar with TOTAL will not need to learn a new vocabulary. The open loop transfer function of the system in Figure II-1 is defined as:

$$OLTF = (GTF)(HTF) \qquad (II-1)$$

where

OLTF is the open loop transfer function

GTF is the forward transfer function

HTF is the feedback transfer function

Similarly, the closed loop transfer function is

$$CLTF = \frac{(GAIN)(GTF)}{1+(GAIN)(GTF)(HTF)} \qquad (II-2)$$

where

CLTF is the closed loop transfer function

GAIN is some real constant

Before the open and closed loop transfer functions can be formed, the forward transfer function, feedback transfer function, and gain must be defined by the user. Since the gain is a real constant, standard techniques can be used to handle this piece of data.

There are two common methods for specifying transfer functions and they are both used by TOTAL and ICECAP. The first method involves specifying the transfer function as a ratio of polynomials. Typically, the user will enter the degree of the polynomials followed by the coefficients. The other common method of defining a transfer function is to specify its polynomial roots or factors. The computer-aided-design tool must allow the user to define transfer functions using either method. When the transfer function is entered in the factored form, the root locus can be plotted directly, but when a transfer function is entered as a ratio of polynomials these polynomials must be factored to determine the roots. Even when transfer functions are

specified by their roots, new denominator roots must be calculated when a closed loop system is formed from an open loop system. Thus, the computer design tool must be capable of factoring large polynomials. This is still a problem for programmers since roundoff error caused by finite word lengths can cause inaccuracies in the factoring.

It can be seen from equations (II-1) and (II-2) that the design tool must also be able to multiply transfer functions. This operation can be accomplished in a number of ways. Again the particular method used depends on how the transfer functions are specified.

Finally, a design tool for conventional control system analysis must be capable of providing both frequency response and time response. The frequency response can be calculated directly from the transfer function regardless of how it was specified (factored or polynomial form). The time response is slightly more complex since an inverse Laplace transform or an approximation of it must be calculated. In addition, the user should be permitted to specify the forcing function (pulse, step, sinusoid, etc.) for the time response.

The above examples serve to illustrate the variety and complexity of mathematical functions required to perform even the most basic control system design and analysis. A complete control system analysis package would require a

comprehensive set of math routines to solve a variety of problems. TOTAL and ICECAP are big steps in this direction.

COMPUTER GRAPHICS

Computer graphics is employed in this work to provide the user with information that is easy to interpret and understand. Studies have shown that recognition of graphical pictures is normally better than alphanumerics and that recall of visual imformation is better than other methods, especially after a time lapse [21:6-7]. With the resolution available in modern display terminals, the computer can draw graphically the root locus, frequency response, and time response. To accomplish the drawing, a basic set of graphic functions is needed. This set of functions is normally called a graphics package and it is usually nothing more than a library of subroutines that provides access to the graphic operations from a higher level language. A good graphics package is general purpose in nature and simplifies the task of writing programs that can run on a variety of systems [25:12]. This ability to create displays for any graphics terminal is known as device independence. The general design of a device independent system is shown in Figure II-2. This device independence allows for portabillity of programs and programmers which the Association for Computing Machinery says is the most important purpose of their proposed core standard [36:2].

Device independent graphics will be a major consideration for this effort.

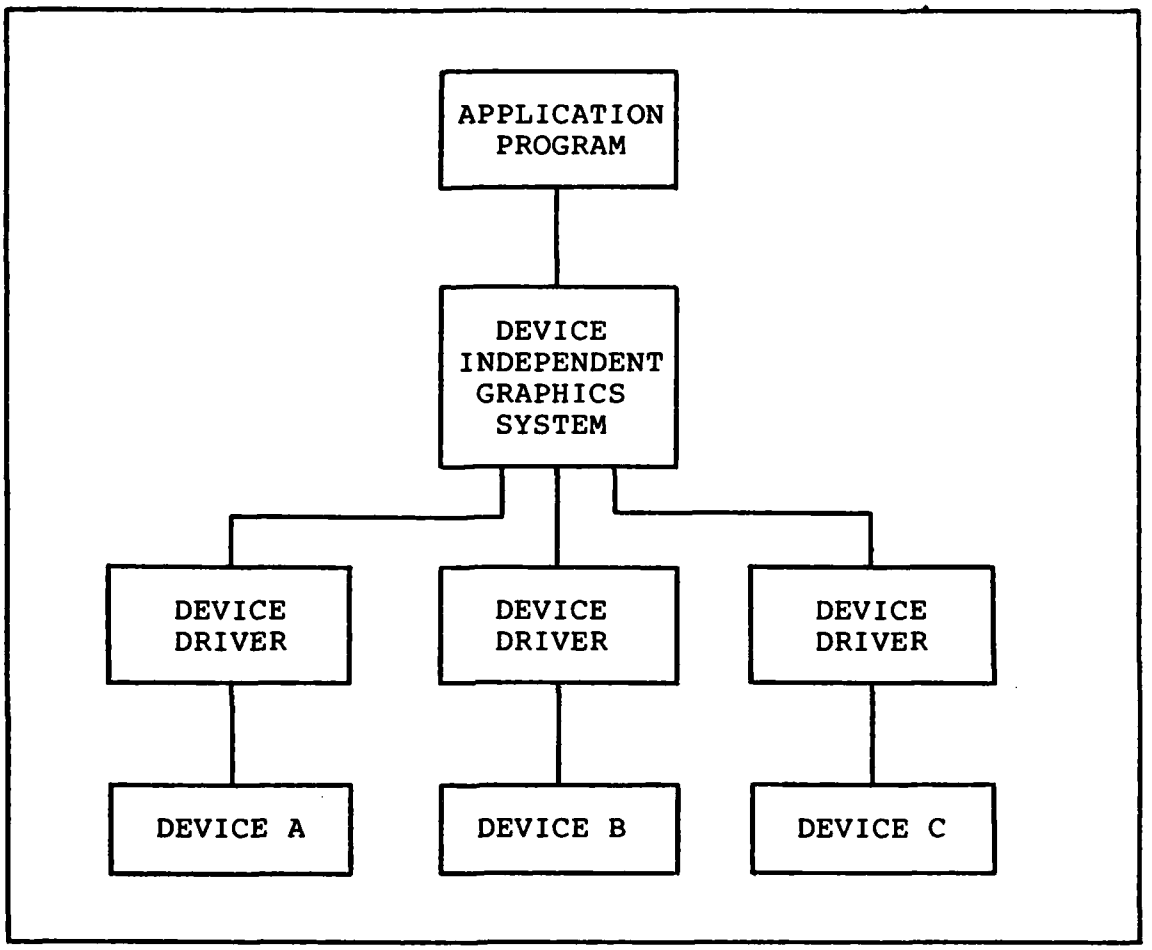


Figure II-2. Device Independent Graphics System [5:II-14]

A graphics system should permit the user to define the picture or portion of a picture that is displayed [25:53]. This type of graphic viewing transformation is known as windowing. Used in conjunction with the window is a viewport which is a rectangular area of the screen where the contents of the window will be displayed. By appropriate

manipulation of the window and viewport, the root locus, frequency response, or time response could be enlarged to fill the entire display. Even further manipulation will permit all three to be displayed simultaneously. A flexible system provides the user with several display options so that only the information desired will appear on the display.

Graphical input techniques can be used to provide information to the computer through a variety of devices such as light pens, joysticks, thumbwheels, and tablets. Normally, the input device is used to move a cursor on the display to a desired location. This would provide an interesting variation on the input of transfer functions: the poles and zeros of a transfer function could be entered graphically on a displayed grid representing the complex plane. However, graphical input lacks the accuracy and precision necessary for control system design but it could provide an easy method for obtaining approximations or observing the effect of modifying a transfer function. Graphical input will not be examined in this effort.

SOFTWARE ENGINEERING

Structured software design is a planning process which involves identifying a variety of relationships between software components. Typical relationships included in structured design are hierarchy of processes, execution

sequences, and transfers of control [27:56]. Many design techniques have been developed and each uses a slightly different method for representing the various relationships.

One such process is the Structured Analysis and Design Technique (SADT) originally developed by SofTech Inc. The basic component of the SADT is the activity diagram or chart. The activity diagram uses rectangular boxes to represent a process or manipulation of data. Arrows then connect the boxes and represent the flow of data and control [27:63]. Several activity charts in a hierarchical arrangement may be necessary to describe an entire system. This type of arrangement is known as the design tree concept or top down design. The basic concept of top down design is that each system or subsystem can be further broken down into a number of smaller independent subsystems.

Another process similar to the SADT is the Systematic Activity Modeling Method (SAMM) developed by Boeing. The SAMM is also a graphical method that has a similar appearance to SADT diagrams with rectangular boxes called activity cells used to identify processes. However, the SAMM is primarily concerned with data flow so the transfer of control is not as apparent as with the SADT.

A third common technique is simply known as structured design. This methodology evolved from concepts developed by W. P. Stevens, G. J. Myers, and L. L. Constantine [27:139].

Unlike the SADT and the SAMM, the structured design method uses three basic tools: data flow diagrams, structure charts, and data dictionaries. The data flow diagram identifies the transformations on the data as it is processed by the system [13:168]. These diagrams are the first step in transforming the conceptual requirements into a final design. The next step is creation of the structure chart which displays the system processes in a hierarchical form. The relationships of the processes or modules are identified along with the flow of data and control between the modules [13:169]. The final stage is the data dictionary which identifies and describes all data elements, processes, and files within the system [19:41].

The structured design methodology allows the designer to begin with the system level data flow and, using the design tree concept, develop a comprehensive and detailed set of data flow diagrams for various levels of subsystems. The designer does not have to be concerned with flow of control at this stage as is required by the SADT. The structure charts then encapsulate the hierarchy developed by the data flow diagrams. These charts provide a wealth of graphical information that is not readily available using the SADT or SAMM. Based on these advantages, the structured design methodology is selected for this effort.

Other requirements of the software design are maintainability, modularity, and documentation. Modularity

and maintainability are closely related. Modularity is actually an off-shoot of structured design which partitions the design into small, logical processes [13:171]. Each module is designed to perform a specific task and it should have little dependence on other modules. Good documentation is also required to have a maintainable system. The tools used in structured design serve to document the design process, but other types of documentation are needed for various purposes. This system will be documented with a user's manual and a source code listing.

USER INTERFACE

The term "user interface" is used to describe the communication between a user and a computer. It is often the most critical part of a design since a good interface can make an application program easy to learn and easy to use [25:13]. On the other hand, a bad user interface can be a source of frustration to the user and eventually lead to user errors and even non-use.

A good user interface should be designed to overcome such psychological factors as boredom, panic, frustration, and confusion [8:251]. Boredom results primarily from lengthy response times. The response time should be consistent with the users view of the complexity of the task. Typically, response times should be less than one

second for simple tasks and less than 15 seconds for complex graphics displays [29:29]. Panic follows boredom when an unusually long delay is encountered and the user may begin to wonder if the system is having problems or the program is stuck in a loop [8:251]. Frustration occurs when the program does not respond as expected and finally confusion occurs when the user is overwhelmed by detail such as a large number of command options or a large amount of information on a single graphical display. Newman and Sproul [25] and Foley and Wallace [24 and 8] have devised similar design processes for the user interface. Newman and Sproul divided the user interface into four components called user's model, command language, feedback, and information display [25:445]. Foley and Wallace divide the design into four similar levels called conceptual, semantic, syntactic, and lexical [24:54]. Green [10] has also examined the general design process and like the other authors, his process begins with a statement of the functional specification which he also calls the mental picture [10:302-304]. By accurately developing a good mental representation, Green contends that the chance of success, or getting a final product that performs the desired functions, is much greater [10:302]. His design process continues with the development of the language which can be compared to Newman and Sproul's command language. Green, however, was not specifically concerned with interactive graphics so he did not consider such topics as

feedback or information display. Thus, Newman and Sproul's terminology which is relatively more descriptive will be used in this effort.

The user's model is a conceptual idea created by the user that relates the functions performed by the program to the data supplied by the user. It is a basic understanding of the process or purpose of the program [25:445]. Newman and Sproul describe the user's mental model as a set of objects and actions. The objects are bits of data that can be manipulated by the actions. "The complete set of actions thus defines the functional capability of the program" [25:448-449]. For this effort, the objects might include a transfer function, a frequency response plot, and a root locus plot. The actions could include multiplying transfer functions, factoring polynomials, and graphical windowing of the display.

The command language is the method emloyed by the user to to tell the computer which action is desired. The commands can be defined easily once the user's model is fully understood. There is a close relationship between the actions of the user's model and the commands which make up the command language [25:449]. Newman and Sproul discuss several styles of command language including keyboard dialogue, keyboard command language, and menu driven command language.

In the keyboard dialogue style, the computer prompts the user for each piece of required data and each action. This style has the advantage of requiring little user training since it requires no memorization of commands. However, it is very rigid and the user may have to answer many questions to perform even simple tasks [25:453-454]. This type of interaction is especially bothersome to the experienced user since the computer is in control rather than the user [1:403].

The keyboard dialogue can be improved by using a keyboard command language. In this method, the user types the entire command which usually consists of a verb and operand. This style is more flexible than the keyboard dialogue but does require a longer learning period. This style is useful for graphics when the keyboard is the only input device available or when the user is required to enter large quantities of numerical or textual data.

One final style to consider is the menu-drived command language in which the user selects the desired action from a displayed list of all possible actions. Once again, the user does not have to memorize commands, but must be able to recognize it on the display [24:46]. One problem with menus is size. As the number of possible commands increases, more and more space on the display is used for the list. In this case, multi-level menus can be used where the first menu selection brings up a new menu [25:456]. However, this can

lead to boredom or frustration when more than two or three menus must be viewed to complete a single command. Green also states that the number of features (such as menu options) should be kept to a minimum since there is a definite trade-off between the number of features and the ease of learning [10:284].

Green's approach to the development of a language is primarily from a semantics viewpoint. He states that it is most important that commands not be ambiguous. The user should know exactly what response will be obtained when a particular command is issued. Likewise, there should not be two or more commands which give the same result [10:285]. Such ambiguous or multiple commands can easily lead to confusion or frustration of the user.

Two other important features of a command language are abort mechanisms and error handling capability. The abort mechanism provides a way for the user to retract unwanted or erroneous commands [25:452]. A good user interface should also provide useful and concise error messages unlike the typical confusing messages received for a compiler error [30:116]. The system should also be forgiving when the user makes an input error. For example, a FORTRAN error (something like "DATA TYPE MISMATCH") is generated at run time when the program is expecting an integer and the user enters a real number.

Displaying the error message leads into the next component of user interface design which is feedback. Immediate feedback can help prevent boredom and panic. If the command was entered properly and the task requires several seconds to execute, the program should provide some type of confirmation that it is still working on the execution [25:465]. One approach is to provide partial answers as they are calculated [8:251]. Another simple way to tell the user that a command was entered properly is to provide an audible beep or tone. The user will quickly learn that the absence of the tone indicates an error [25:466]. As discussed in the preceeding paragraph, rapid response is needed to an input error. The error message should make clear the cause of the error and indicate corrective action. It is really a type of computer aided instruction [14:263-264]. Finally, for users continuing to have problems, the program should permit on-line help and even an on-line tutorial if necessary [30:116].

The final component of the user interface for a graphics system is the information display. This component is concerned with the most effective method of displaying information [25:459]. The best display should provide the minimum amount of information required by the user's model. Providing more information than necessary can lead to confusion. Human users are only capable of recognizing a limited number of colors, shapes, intensities, and line

styles [8:252]. Screen space must be utilized efficiently and a flexible graphics system permits the user to rearrange the display to suit his personal tastes [25:461]. A final point to consider for information display is the quality of the image. The graphics display should use symbols familiar to the user. For example, curved lines are usually constructed from a number of straight line segments [25:462]. A curve will appear smoother to the user if very small segments are used while a curve drawn with long straight segments may distract the user and lead to frustration or confusion. "Care devoted to the selection of information display techniques pays off handsomely to its effect on the overall quality of the user interface" [25:463].

SUMMARY OF REQUIREMENTS

The complete set of functional requirements is summarized in Table II-1. Each entry is assigned a priority ranging from one to three. Those functions having a priority of one are considered to be the minimum essential requirements of an interactive control engineering program. Demonstrating these basic functions generally shows that all system analysis tasks can eventually be performed with computer graphics. Successful implementation of the priority one features also confirms the validity of the design process and programming strategy. Finally, all

continuous time functions are included in priority one and discrete time functions are included in priority two primarily because this is the order in which they are usually taught. If successful, the interactive graphics system could be placed into immediate use for students learning system analysis. As a minimum, all priority one requirements are considered in this effort.

1.	Root Locus Analysis (1)
2.	Continuous Time Response (1)
3.	Continuous Frequency Response (1)
4.	Transfer Function Manipulation (1)
5.	Polynomial Operations (1)
6.	Discrete Time Response (2)
7.	Discrete Frequency Response (2)
8.	Continuous to Discrete Transformations (2)
9.	Matrix Operations (2)
10.	Scalar Operations (scientific calculator) (2)
11.	Filter Design (3)
12.	Compensator Design (3)
13.	Stochastic Estimation & Kalman Filter Design (3)
14.	Stability Analysis (3)

Table II-1. Functional Requirements and Priorities

Discrete time functions can then be added as time and resources permit. Thus, the priority two requirements are those functions which complete the set of conventional

analysis and design techniques. Finally, priority three is composed of those functions required for advanced design and stochastic analysis.

SUMMARY

This chapter has defined and discussed those requirements considered to be essential for computer aided control system design and analysis. The requirements are categorized into four areas. The first area, system analysis, covers all functions and operations to be performed and thus defines the functional requirements. The remaining areas (computer graphics, software engineering, and user interface) are considered general design requirements which are aimed at producing a computer aided design tool which is easy to learn, easy to use, and easy to maintain.

CHAPTER III

SYSTEM DESIGN

INTRODUCTION

This chapter will cover the issues which must be considered in the design of a graphics program for control system analysis. Decisions will be made in selecting the graphics package, choosing a programming language, and developing the user interface. This chapter will also discuss the design structures of TOTAL and ICECAP and how they relate to the system design.

GRAPHICS SOFTWARE

The selection of graphics software can have a major impact on the system design. The next few paragraphs include a discussion and comparison of the various device independent software options. Finally, considering the constraint of available software, a graphics package will be selected.

Before evaluating graphics software, it is important to understand the basic terminology. The three most significant functional components of graphic software are primitives, attributes, and viewing transformations.

The primitives are the basic geometric shapes used to

create graphical pictures. Typically, the set of primitives includes points, lines, text, and polygons [25:82]. By appropriately manipulating these basic shapes, nearly any two or three dimensional display can be generated. Since text is frequently used in conjunction with the graphical drawings, the alphabet is usually included in the set of primitives.

Associated with each primitive is a set of attributes which are basically the features or characteristics of the primitive. For example, some common attributes of the line are width, color, style (solid, dotted, or dashed) and intensity. Attributes for text include color, font, size, and spacing. There are also general attributes which apply to all primitives that permit scaling, rotation, and translation [29].

The final important function is the viewing transformation which permits the programmer or user to select any rectangular portion of a graphical picture and reproduce it on any rectangular area of the display surface. The rectangular portion of the picture is called the window and the rectangular portion of the display surface is called the viewport [25:74]. A process known as clipping is used to discard or "make invisible" those portions of the picture outside the window.

With the rapid evolution of computer graphics in the

1970's, the need for an industry wide standard became apparent. This standard would define all the functional requirements for graphics software - a standard set of primitives, attributes, and viewing transformations would be established. In cooperation with the National Bureau of Standards and the American National Standards Institute (ANSI), the Association for Computing Machinery established a Graphics Standards Planning Committee to prepare a draft standard [36:2]. The committee's first proposal was delivered in 1977 and has become known as the Core System. A detailed revision of the standard [36] was published in 1979 and although it was never adopted by ANSI, it was implemented by several colleges, laboratories, and commercial vendors [16]. In the late 1970's, the focus changed to a world wide standard and the Graphical Kernal System (GKS) developed by the West German Standards Institute (DIN) was accepted by the International Standards Organization as a draft international standard [2]. In October, 1982, ANSI voted to begin necessary action to approve GKS as an American National Standard [2]. This action resulted primarily from ISO's acceptance of GKS. The GKS could be adopted as early as 1984.

The GKS and Core are similar in many respects but there are two significant differences. First, the GKS is a two dimensional system, while the Core has both two and three dimensional primitives and transformations. Second, the GKS

has only absolute primitives while the Core has both absolute and relative primitives [5:28-26]. Although neither system is bound to any specific language, it should be noted that ANSI has voted to include a FORTRAN binding of GKS as an appendix to the American Standard [2].

In AFIT's Digital Engineering Lab, three graphics packages have received considerable study and use. The first package is PLOT-10, a Tektronix commercial product developed for use with the Tektronix 4010 family of displays [28:2]. PLOT-10 is used extensively by TOTAL and VAX-TOTAL to produce high resolution displays of system responses [19:231]. Since PLOT-10 is designed for the Tektronix displays, it is not a device independent package. Even though many modern graphic displays have a Tektronix emulation mode, PLOT-10 does not permit the programmer to use many of the non-Tektronix features and capabilities. For these reasons, PLOT-10 will not be used for this effort.

One of the first AFIT efforts to acquire a comprehensive device-independent package was conducted by Philip Tarbell in 1981. He reviewed several packages but the majority of his work centered on obtaining a package known as GRAFLIB which was developed at the Lawrence Livermore Laboratory. GRAFLIB is a subroutine library intended to be used with FORTRAN application programs - the GRAFLIB source code is actually LRLTRAN [15:3]. GRAFLIB is divided into three basic levels that permit a variety of

applications and programmer interfaces. The lowest level of GRAFLIB is a basic set of graphics functions called GRAFCORE. These are the elementary operations used to build the higher levels of GRAFLIB. The intermediate level provides a variety of higher level routines for such operations as plotting grids, arcs, circles, and rotating 3D objects. The highest level contains routines for drawing entire pictures. At this level, a single subroutine call will draw an entire graph with grids, labeled axis, and a title [15:3]. GRAFCORE is based loosely on the Core System but most of the functions in the intermediate and upper level of GRAFLIB have no counterpart in the Core.

Because of the large amount of available code, Tarbell elected to transfer only the subset of functions that most resembled those in the proposed Core Standard [37:98]. After numerous problems, Tarbell successfully transferred a number of functions to AFIT's VAX. This package contained only output functions and lacked a device driver. Tarbell finally recommended that GRAFCORE be used for graphics research and development and that a more direct implementation of the Core System be acquired to study graphics applications [37:120].

The package recommended by Tarbell is known as the GWCORE and it was successfully used at AFIT in 1982 by Kevin Rose [29]. The GWCORE is a product of George Washington University and is a direct FORTRAN implementation of the

Core System. There is a one to one correspondence between the GWCORE subroutines and the specified Core System functions [36 and 38]. Although this was a strict implementation of the Core standard, it was incomplete since hidden surface removal and the input functions were not included. However, input functions based on the Core standard had been developed at AFIT in 1980 by Harold Curling [6]. The GWCORE proved to be a very useful package since the documentation [36, 38, and 43] was concise and complete. GWCORE was developed on a VAX thus making the source code easy to transfer and eliminating any need for language translators or pre-compilers which caused problems in the transfer of GRAFLIB. The GWCORE was also supplied with a device driver for the Tektronix 4014 display. This allowed immediate testing, experimentation, and development of applications programs. Other device drivers could be developed as needed.

An alternative to the graphics package is a graphics language in which there are special commands or statements for creating pictures. Some examples of commands might be draw, point, circle, and paint. Such graphic languages have not come into widespread use because they require their own unique compiler or interpreter and are often difficult to use in conjunction with other high level languages [25:436].

Based on the above comparisons, the GWCORE is best suited for this system analysis application. The completed

user and design documents along with numerous studies and data on the Core System should make the final product easy to transfer to other machines and easy to understand by other programmers. Even though the use of higher level routines in GRAFLIB could make this effort very simple, it is felt that the GWCORE, which is a stricter implementation of the Core standard, has more long range value. Finally, the availability of a proven device driver confirms the GWCORE as the proper choice.

SELECTING A PROGRAMMING LANGUAGE

Selecting a high level language for the development of application programs is an issue that frequently does not receive the consideration that it should. Inexperienced programmers will often use the language which is most familiar to them rather than investigate a new or different language. Choice of a programming language can have a large impact on algorithm design and data structures [19:25]. The most important consideration in choosing a language is the nature of the problem to be solved. It is well known that certain types of problems can be handled best by some particular language. Each language has distinct advantages and disadvantages and the trade-offs must be evaluated always keeping in mind the nature of the problem. Although there is a variety of high level languages in use today, this discussion will be limited to the languages available

on the AFIT VAX; FORTRAN and Pascal.

FORTRAN is the oldest of the two languages having been developed by IBM in the 1950's. FORTRAN has undergone many changes since then and today it is probably the most popular language for scientific and engineering applications [26:308]. Just as there is a need for a device independent graphics standard, the FORTRAN user community desired a machine independent standard that would improve the portability of programs. ANSI issued such a standard in 1966 and a major revision in 1977 but most versions of FORTRAN still contain various extensions that are machine dependent. The structure of FORTRAN consists of program units. There is nearly always a main program along with any number of subprograms. The subprogram can be either a subroutine or a function. Unlike Pascal, the FORTRAN subprograms are not nested in a hierarchy inside the main program [26:309]. Thus FORTRAN does not readily support a direct translation of the structured design into a structured program [19:31]. It is possible though to still use a modular approach and to develop software modules that have little dependency on other modules. On the other hand, FORTRAN contains a much larger set of math functions that allow programming of complex engineering and statistical problems. With early versions of FORTRAN, manipulation of text or character strings was difficult if at all possible, but the 1977 standard provided a character

variable type and several character operations [26:340].

Pascal is a relatively new language having been developed in the 1970's. Pascal is well known for its data handling capabilities and structured programming style [26:560]. Pascal derives its power from user defined data types such as records, arrays, and sets. The structure of a Pascal program is quite different from that of a FORTRAN program. A Pascal program contains a heading and a block. The heading contains the name of the program and typically some file information. The block contains a declaration section and an executable section [13:1-6]. Pascal requires that all variables, constants, data types, and subprograms be declared. Two types of subprograms are used (functions and procedures) and they are nested within the main program or other subprograms. Because of Pascal's structure and English-like statements, programs are easy to read and understand [41:1-1].

A comparison of numerical precision and range is also necessary considering the nature of the problem. ICECAP allows users to define polynomials having an order of 50 or less. Evaluating such a transfer function over a large range of frequencies can result in large magnitudes and even overflow. Increased ranges available with some languages can be used to reduce the severity of this problem. Likewise the increased precision offered by some languages can be used to improve the accuracy of the system analysis

computations. The basic real variable in VAX-11 FORTRAN and VAX-11 Pascal uses four bytes (32 bits) for storage and has a range in magnitude from 0.29E-38 to 1.7E38 [40:2-7 to 2-9 and 42:2-2]. The VAX-11 Pascal stops here but FORTRAN has several additional options. VAX-11 FORTRAN provides an eight byte variable which can be used in two ways. One type allows the same range of magnitude as the four byte variable but more significant digits are used thus providing greater accuracy. This is the double precision variable. The second type of eight byte variable has similar precision to the four byte variable but an expanded range of magnitude. Finally, VAX-11 FORTRAN allows programmers to use sixteen byte real variables having a range of magnitude from 0.84Q-4932 to 0.59Q4932 [40:2-7 to 2-9]. It should also be noted that VAX-11 Pascal has no convenient method for handling complex variables which are essential in system analysis. The VAX-11 FORTRAN provides four types of complex variables for the user.

On the Digital Equipment Corporation VAX, it is possible to link FORTRAN and Pascal programs, thus obtaining the advantages of both languages. All FORTRAN subprograms to be referenced from Pascal programs or procedures must be identified as "external" in the declaration section. This was successfully accomplished with ICECAP where Pascal was used to decode and interpret commands and control overall flow of the program, while numerous FORTRAN subroutines were

used to perform the mathematical functions. Machines not having compatible compilers for both languages will not permit this link and doing so may create a machine dependency for the VAX.

Another consideration for this effort is the interface to the GWCORE graphics package. Although the GWCORE is a FORTRAN implementation, the subroutines can be accessed from either Pascal or FORTRAN as discussed in the preceeding paragraph. However, the GWCORE contains numerous routines and variables and the declaration of all of them in a Pascal program could be quite lengthy and cumbersome. Another problem discussed in the Core proposal is the intermixing of program language I/O and Core System I/O to the same device [36:II-105]. The Core proposal does not guarantee that this will always work and in fact such a problem has been encountered at AFIT when mixing a FORTRAN READ with the GWCORE output. The problem appears to be extra line feeds which causes a confusing distortion to the picture. George Washington Univesity has documented this problem [38:37] and provided a solution for FORTRAN users but makes no mention of a problem or solution when using Pascal I/O.

The final consideration in selecting a programming language is the availability of existing code that can be used or modified for this effort. A tremendous amount of code exists (from ICECAP) in both languages. It is desirable to use as much of this as possible since the

FORTRAN math routines have been used for several years and have proven to be reliable and accurate. The Pascal portion of ICECAP is a sound foundation for adding new commands and functions. By using as much of the existing software as possible, the resulting graphics program will have a flow and appearance similar to ICECAP, thus making it easy to learn for those already familiar with ICECAP.

Based on the above comparisons, it is most advantageous to actually use both languages. This will continue the programming concept developed in previous efforts by Logan and Gembarowski. Despite a possible machine dependency for the VAX, it is believed that using each language where it is best suited and making full use of the available code is a greater benefit.

THE USER'S MODEL

A definition and discussion of the user's model was included in Chapter II. Even though developed by the programmer, the user's model represents the user's (not the programmer's) understanding of the program's data elements and the manipulations that affect those pieces of data. This is very similar to information that is contained in top level data flow diagrams but the user's model is really more concerned with manipulations and transformations to the data rather than its flow through the system. The data flow

diagrams represent the programmer's or designer's view of the system.

There are a number of ways in which the user's model can be developed. Larimer's approach in developing the program TOTAL was to actually begin writing the user's manual in the early stages of the design [17:4]. Newman and Sproull discuss another useful method of preparing the user's model. Their method consists of listing a set of objects (data elements) and the associated set of actions (manipulations) the user can apply to those objects [25:448-449]. This method is not only very simple but it is also very useful in developing the commands which make up the system's command language. Table III-1 illustrates the user's model for the continuous time functions to be implemented in this effort. These objects and actions are derived from the discussion of functional requirements in Chapter II as well as from a detailed study of continuous time functions performed by TOTAL [18] and ICECAP [9]. The next section of this chapter demonstrates how the objects and actions are transformed into commands.

THE COMMAND LANGUAGE

The command language is the method in which the user expresses his desired action to the computer. Three common methods were discussed in Chapter II. Of the three methods, the most preferable are the keyboard command language and

TRANSFER FUNCTIONS

OBJECTS	ACTIONS
HTF	define, copy, display, print
GTF	define, copy, display, print
OLTF	define, copy, display, form, print
CLTF	define, copy, display, form, print
GAIN	define, display

RESPONSES

OBJECTS	ACTIONS
Root Locus	display, print
Time Response	display, print
Frequency Response	display, print
Figures of Merit	display, print
Forcing Function	define

DISPLAY ATTRIBUTES

OBJECTS	ACTIONS
Closed Loop Mode	turn on, turn off
Grids	turn on, turn off
Decibel Mode	turn on, turn off
Radian Mode	turn on, turn off

Table III-1. User's Model for Graphics Application Program

the menu-driven command language. The keyboard dialogue is the least desirable since it has a very rigid structure and flow with the computer really having control rather than the user. Gembarowski developed an interesting combination of the first two methods which will be continued in this effort. Before this method is presented, a brief discussion of command formulation is in order.

Green has examined the English language from a grammar standpoint and found that the basic structure consists of actor, action, object, and modifier. When an English sentence is parsed by the reader, it is found that these four items contain the majority of the information [10:294]. The basic point being made by Green is that computer languages (both command languages and programming languages) should be similar to the user's natural language. Therefore, English like commands which should be easiest to learn and understand should be formulated from the basic structure of the English language. Keyboard commands consist of four similar elements with slight modification to accomodate the requirements of the computer and programming language [25:454]. The first element is a verb which is usually the same as or similar to the action found in the user's model. The second element is an operand to which the verb applies. As one might expect, the operand is normally the object of the user's model. The third element is a modifier which provides instructions on how to interpret the

verb and operand. An example would be the two ICECAP commands:

```
DEFINE OLTF FACT
DEFINE OLTF POLY
```

Both commands specify that an open loop transfer function is to be defined, but the modifier to the first command states that it will be defined in factored form while the modifier in the second command specifies that the OLTF will be defined in polynomial form. Commands such as these are actually incomplete if the modifier is not included. Commands may contain more than one modifier even though none of the modifiers may be explicitly shown in the user's model. The fourth element of a command is a set of delimiters used to separate the verb, operand, and modifier. A blank space is used as a delimiter in the commands shown above. The carriage return is also considered a delimiter when it is used to signal the end of a command. With the above definitions in mind, some of the possible commands available from the first line of the user's model (Table III-1) are:

```
DEFINE HTF FACT
DEFINE HTF POLY
COPY HTF GTF
DISPLAY HTF
PRINT HTF
```

Note that the first three commands have modifiers which did not appear in the user's model but are derived from the

functional requirements discussed in Chapter II. As can be seen above, it is not necessary for commands to contain all four components. It is sometimes practical to use only a verb. The complete set of commands for this particular project can be found in the user's manual contained in the appendix.

The discussion will now return to Gembarowski's command language which is a combination of keyboard command language and menu-driven command language. The user is initially presented with a list (menu) of all the valid verbs. If the user responds with a complete command the requested action is taken. However, if the user enters only a verb and it is not a complete command, he is presented with a list of valid objects for that verb and permitted to abort or complete the command. This process is also used where modifiers are needed. Since this command language satisfies the criteria of a good user interface and the source code is readily available, it is selected as the command language for this project. Even though many modifications and additions will be necessary, the development of an entirely new command language would be wasteful duplication.

INFORMATION DISPLAY

The concept of information display is concerned with the optimum display of program results to the user. Graphics applications programs frequently divide the screen

into three areas. One area is used for prompts and error messages, the second area is used for a command menu, and the third area is used to display application program data or results [25:461]. Kevin Rose used this approach in the design of his interactive graphics editor [29]. In the application of control system design, it is beneficial to view and study several characteristics at the same time. Thus, several display options are created to display simultaneously such information as frequency response (magnitude and/or phase), time response, root locus, and the system's figures of merit. These display options are basic extensions to the user's model that improve the information display and therefore create a better user interface. Placing four plots, even on a large display, results in relatively small charts so the user will also be provided with commands that permit any single plot to be magnified to fill the entire display.

The number of display options will be kept to a minimum to avoid the possibility of creating confusion. Also, because of the large amount of application data to be displayed, the screen will not be divided into three areas as suggested at the beginning of this section. Basically two screens will be included in the interactive display. The first screen will provide the user with a menu and prompts. Once the user responds with a command the first screen will be erased and a new screen created to display

the application program results. This approach may seem undesirable to some, but four plots, a command menu, and a prompting area is an overwhelming amount of information to "squeeze" onto a single display. The user will be permitted to exit from the data display and return to the menu display by striking a single key. Such a display method should cause little or no degradation to the overall user interface.

DESIGN STRUCTURE OF TOTAL

Before discussing the design structure of the planned graphics system it is helpful to discuss briefly the structures of TOTAL (actually VAX TOTAL) and ICECAP since they play such a big role in the new design. This section will discuss the basic design ot TOTAL and the next section will cover ICECAP.

TOTAL is a relatively large collection of FORTRAN subroutines that is capable of performing a variety of system analysis functions. A top level structure chart for TOTAL is shown in Figure III-1. The command language of TOTAL is similar to a menu-driven system but there are really no commands. The user is required to enter an option number which corresponds to the desired operation. The verbs and operands are implied by the option number. The option number is used in conjunction with a massive computed go-to statement to determine which of the first level

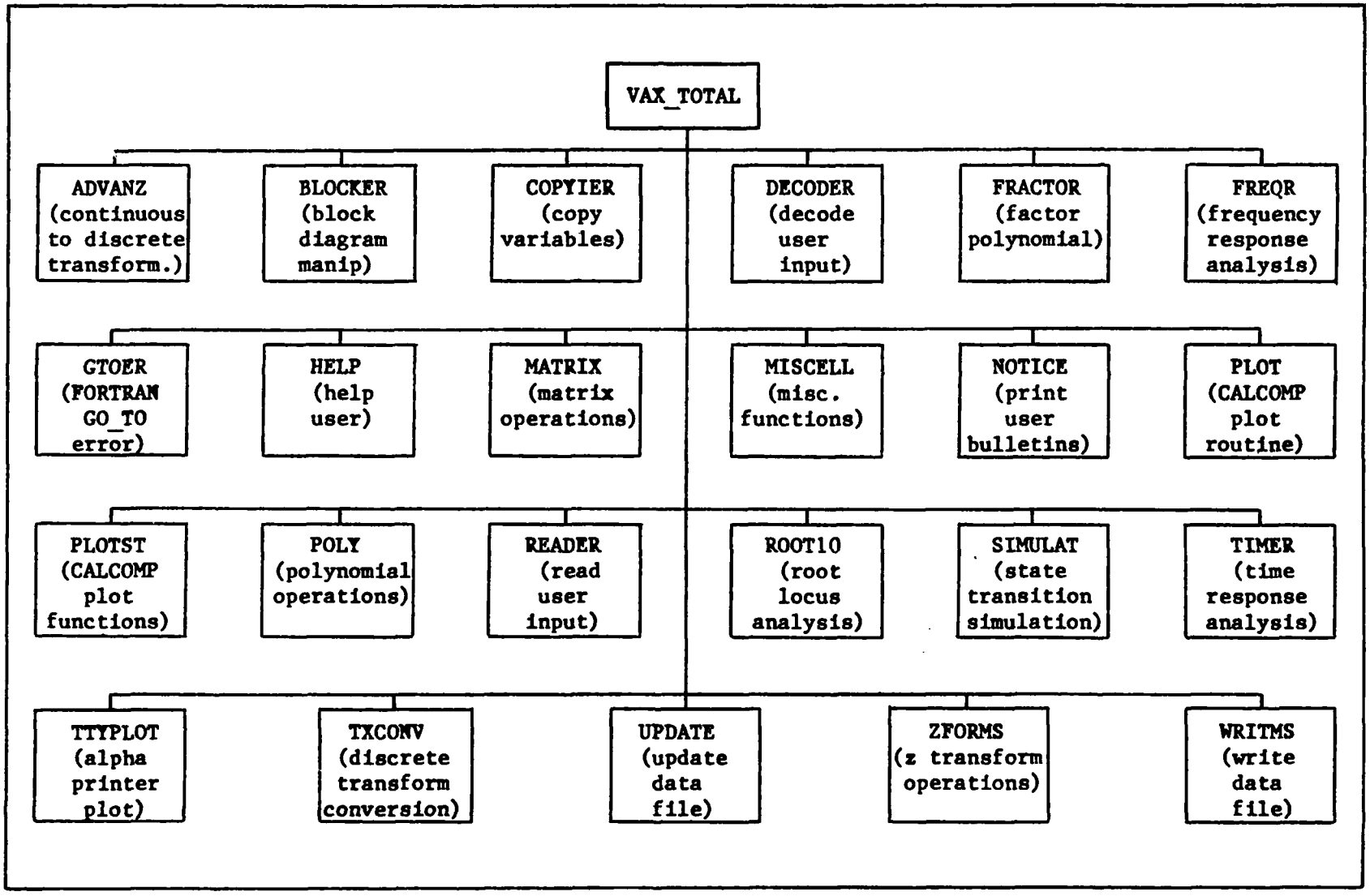


Figure III-1. Top Level Structure of VAX_TOTAL

subroutines is called. These subroutines can be seen in the structure chart. For some TOTAL operations, bits of data are entered on the same line as the option number. These data items could be considered operands or modifiers without verbs and are interpreted by the subroutine READER before the selected option is executed. The subroutine READER also provides protection against FORTRAN input errors. Finally, there are instances where data can be entered without selecting any option. For example, to set the forward gain to 100, the user types "GAIN=100.0". This variety of input modes can be difficult to learn and it is expected that the command language concept used for ICECAP and this effort will make the new system analysis packages easier to learn and use.

DESIGN STRUCTURE OF ICECAP

ICECAP can be considered as an extension to TOTAL for the purpose of improving the user interface. The top level program of this system is now ICECAP with an associated set of Pascal procedures used to interpret the commands and provide various control features for the Digital Equipment Corporation VT-100 display. ICECAP has a command language very similar to the one discussed earlier in this chapter and many of the commands developed from the user's model and the functional requirements will be identical to those in ICECAP. The commands for the various display options have

no counterparts in ICECAP.

Figure III-2 shows the top level structure chart for ICECAP. In simple terms, the basic function of ICECAP is to read the user's command and convert it to the appropriate option number which is then passed to TOTAL through the procedure INTERPRET. This was implemented by changing the main program TOTAL into a subroutine called TOTICE. Data can also be passed to TOTAL in the same way. By mentally placing the structure chart of TOTAL under the procedure INTERPRET (fig III-2), it is easy to visualize the relationship between TOTAL and ICECAP.

DESIGN STRUCTURE OF ICECAP-II

ICECAP-II is chosen as the name for the proposed graphics system since the new design is based heavily on the design of ICECAP. In addition the new program will have a structure, appearance, and general flow similar to ICECAP.

The structured design of ICECAP-II begins with the development of data flow diagrams. These diagrams represent the basic data elements (objects) and operations (actions) that make up the program. The diagrams are derived primarily from the functional requirements and are the first step in the implementation of a modular system. For this effort, data flow diagrams were developed for the entire set of continuous time functions even though many of the

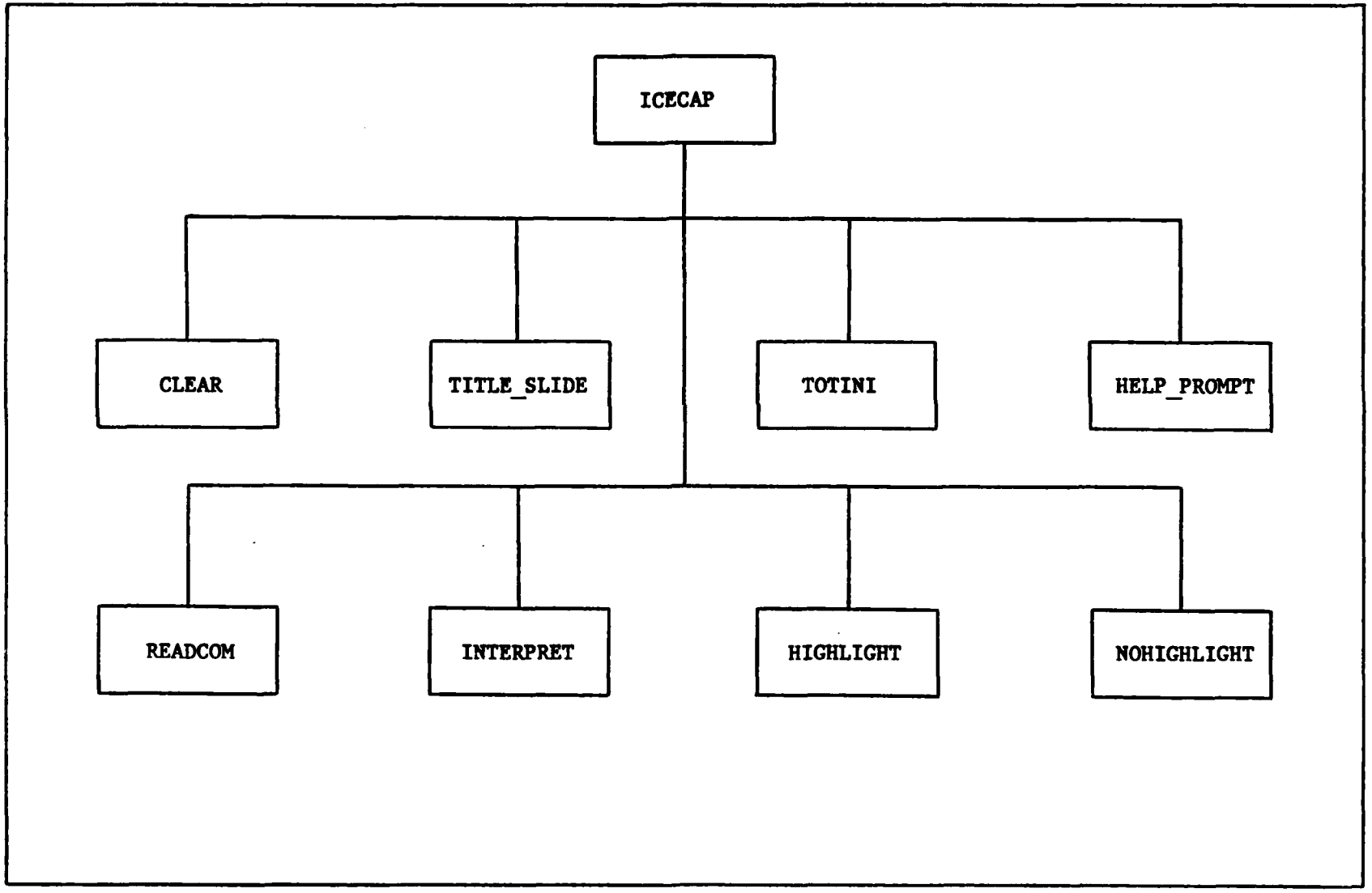


Figure III-2. Structure Chart for Program ICECAP

functions will be performed with the existing ICECAP software. The advantage of preparing a complete set of diagrams lies in its assistance in identifying those areas which must be modified and it particularly helps to locate the proper points to interface the application program to the GWCORE. Preparation of the diagrams also results in a good understanding of ICECAP and its structure. The top level diagram is shown in Figure III-3 and a complete set is included in Appendix A. The nodes of the system are labeled:

1	Format Input
2	Execute Command
2.2	Evaluate Error
2.3	Perform System Analysis
2.3.2	Update Data File
2.3.2.3	Enter Transfer Function
2.3.2.4	Enter Transfer Function
2.3.2.5	Form OLTF & CLTF
2.3.3	Perform Conventional Analysis
2.3.3.2	Calculate Root Locus
2.3.3.3	Calculate Frequency Response
2.3.3.4	Calculate Time Response
2.3.4	Set Graphics Parameters
2.4	Determine Help Needed
2.4.2	Provide Command Description
2.5	Format Response
3	Perform Graphic Functions
3.1	Establish Windows & Viewports
3.2	Display Text
3.3	Display Data

CONCLUSION

Several important design decisions have been discussed in this chapter. A comparison of programming language trade-

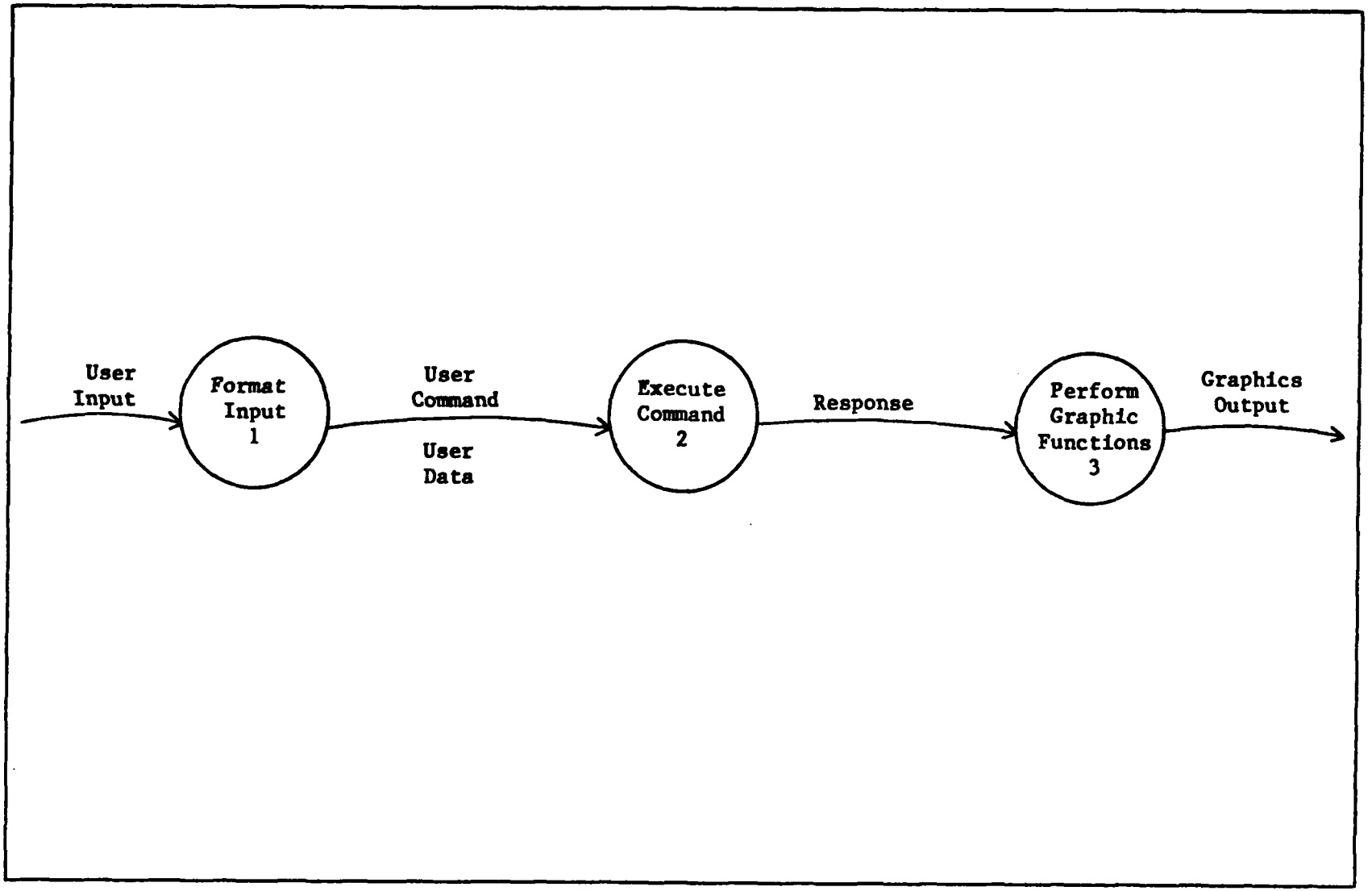


Figure III-3. Overall System Diagram

offs and a review of available software concluded with the selection of both Pascal and FORTRAN for the implementation. A verb-operand-modifier type of command was selected for the command language and the method of deriving the commands from the user's model was outlined. Finally, the design structure of existing programs was discussed and related to the design of ICECAP-II. This structured methodology will be used throughout the implementation and discussed further in the next chapter.

CHAPTER IV

SYSTEM IMPLEMENTATION

INTRODUCTION

This chapter covers the important considerations in the implementation of ICECAP-II. The chapter begins with a general discussion of the strategy and approach followed by a more detailed examination of how the interactive graphics was interfaced to the existing software. The next section looks at the display options included in the implementations and the method in which they were developed. Following the topic of display options is a discussion of problems encountered with the GWCORE as well as solutions. The last topic of the chapter considers the potential application of array processors to both graphics and system analysis.

IMPLEMENTATION STRATEGY

The selection of an implementation method can have a significant impact on both the difficulty and success of the actual coding. The most popular coding method is the top-down approach [44:340] but many programmers prefer what is known as bottom-up programming. These techniques refer to the software hierarchy found in structured design. The only difference is the order in which the coding takes place. In the top-down strategy, the modules at the top of the

hierarchy are coded first while the bottom-up implementation begins coding with the modules at the bottom of the hierarchy.

Yourdon and Constantine discuss two other approaches that have proven useful in certain situations. The first method discussed is called a phased implementation. This method consists of three major steps: (1.) code and test all modules individually, (2.) integrate all modules, and (3.) test the system [44:341]. The major disadvantage of this method is the difficulty in debugging when each module has passed its test but the system fails [19:75]. Since all modules are integrated at the same time, the programmer has no logical point to begin troubleshooting. For these reasons, the phased implementation is most useful when the system is composed of a small number of extremely independent modules. The phased implementation is frequently, but not necessarily, associated with the bottom-up implementation since in the bottom-up method several modules must usually be coded and integrated before testing can be accomplished.

The other method discussed by Yourdon and Constantine is called the incremental implementation. This method can also be summarized in three basic steps: (1.) code and test a single module, (2.) add another tested module to the first and test the resulting combination, and (3.) repeat this process until the entire system has been developed and

tested [44:341-342]. If a problem is encountered, the debugging obviously begins with the last module integrated into the system. Even if the bug is not located in this module, at least the programmer has a logical starting point. As one might expect, the incremental implementation is often associated with top-down design.

In reality, it is often necessary to compromise and either combine or modify the implementation strategies discussed above and such is the case for ICECAP-II. Although a top-down structure or design tree concept is being maintained for ICECAP-II, a complete top-down implementation is not practical since such a large amount of software has been coded and tested in the developments of ICECAP and VAX TOTAL. The incremental approach is selected as the primary strategy for ICECAP-II. Each module added is coded and tested before being integrated into the system. The modules are added one at a time with a complete system test accomplished before adding the next module. Modifications to the existing software are performed in the same way. A single modification will be incorporated and tested before proceeding to the next modification. By maintaining this approach throughout the implementation and by working out all bugs before proceeding to the next module or modification, a relatively trouble free system and early success is expected.

INTERFACE TO ICECAP

Incorporating interactive graphics into the existing ICECAP software was accomplished in two major phases. Phase one involved removing all device dependencies from ICECAP while phase two was concerned with adding the interactive graphics. This logical approach can be used in adding graphics to nearly any existing program.

ICECAP contained a significant number of dependencies for Digital Equipment Corporation's VT-100 video display. Most of these consisted of escape or control sequences which performed such functions as clearing the screen, displaying in inverse video, moving the cursor, and using the VT-100 graphic character set. All of these dependencies have to be removed or converted to an equivalent Core system operation. For example, the escape sequence to clear the screen can be replaced in many cases by the Core system's new frame action. If retained segments are displayed, the new frame action is activated by deleting any or all of the segments. Portions of code using the VT-100 graphic character set are replaced with Core system line drawing functions. Gembarowski uses the graphic character set of the VT-100 to draw a set of control system block diagrams when an invalid command is issued by a user performing transfer function manipulations. In this case, all of the VT-100 dependent code (in PASCAL) was eliminated and replaced by a call to a FORTRAN module which uses the Core system to display a

similar set of block diagrams. The Core generated output is shown in Figure IV-1. This is an excellent example of how graphics can be used to provide assistance to the user who is learning control system design or to a more experienced user who cannot recall the exact command.

An interesting application of computer graphics was demonstrated when developing this display. The basic geometric shapes required for the block diagrams are lines, rectangles, circles, and arrows. However, the Core system uses the line as its basic primitive. The application programmer must then create the more complex shapes using the appropriate combination of lines. In a case such as the set of block diagrams where special figures are used repeatedly, it is convenient to create a higher level module for each shape which then makes the necessary call to the Core system. These high level routines are similar to the intermediate level of GRAFLIB which was discussed in Chapter III. Establishing this "library" of commonly used shapes simplifies the coding process and also makes the shapes easy to use in other parts of the program.

The second phase of the implementation is concerned with the use of graphics for the ICECAP outputs such as the time and frequency responses. VAX TOTAL, which is the foundation of ICECAP, provides output in a variety of formats including tabular listings, alpha-numeric character plots, CALCOMP plots, and even a few graphic outputs using

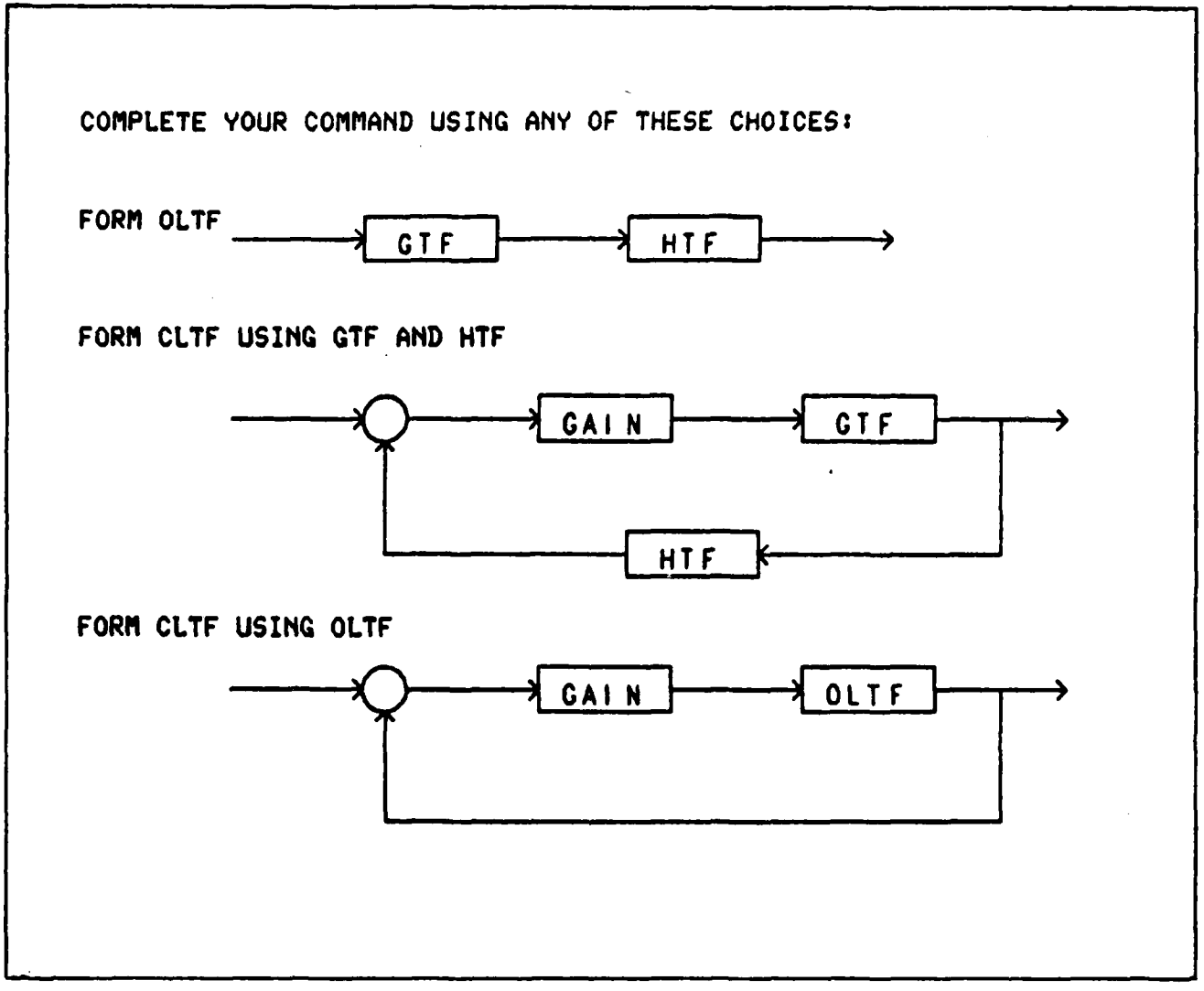


Figure IV-1. ICECAP-II's Response to an Invalid "FORM" Command

the Tektronix PLOT-10 package [18].

The first consideration in adding graphics to the existing software is the ease in obtaining the data to be plotted as well as its format. Clearly, from the preceding paragraph, the data is readily available in a tabular listing so this feature was used for the time and frequency response plots. A slightly different method was used for the root locus. It will be disucssed in the following paragraph. The only modification required to the tabular listings is the number of data points generated. Typical tabular listings produce less than 50 data points which is insufficient for creating a high resolution graphics plot. The appropriate modules are thus modified to provide a more suitable set of data: 400 points for the time response and 90 points per logarithmic cycle for frequency responses. In each case (except for the root locus) the data points are stored in an array, passed to a scaling routine and then passed to the Core system's polyline function for plotting. The module used to scale the data is also used to perform a few related funtions such as labelling the axes. The only disadvantage of using the tabular data listing in this way is that it takes the feature away from the user. The tabular listings are useful for examining a time or frequency response around a particular point. However, the main objective of this effort is to demonstrate the use of standard device independent graphics for system analysis and

the tabular listing is the most obvious place to join ICECAP with the GWCORE. Following a successful implementation, the tabular listing feature can be restored and the original alpha-numeric character plots can be replaced by the new graphic plots.

The root locus plot is handled slightly different. The primary difference is the use of a disk file rather than an array for storage of the data points. This method was selected after examining the existing root locus software and noting the following points which make array storage difficult:

(1.) The number of locus traces is dependendent on the problem and each trace would have to be stored in a unique array to simplify using the Core system's polyline function.

(2.) The length of each locus trace is dependent on the nature of the problem thus making array size hard to determine.

(3.) The actual data point can be calculated at any one of three points in the root locus module. Keeping track of array pointers at each place is difficult.

By using a data file each point can be written to the file as it is calculated. The traces can be separated in the

file by some non-numeric character (a "$" in this case). Use of a data file may cause a slight reduction in execution time but it does eliminate a substantial amount of bookkeeping since the length and number of traces is inconsequential. Until the root locus programs can be re-examined or rewritten it is felt that the file storage should continue to be used.

DISPLAY OPTIONS

Although the various display options were included in the design from the beginning of this effort, they were not implemented until each of the basic displays had been coded and tested. That is, the time response, frequency responses (magnitude and phase), and the root locus were developed individually before any attempt was made to display two or more simultaneously. By implementing each display individually, it is then a simple matter of viewport manipulation to display any two or more.

Identifying the display options to be included in ICECAP-II is a matter related to the user interface. Both the concepts of the command language and information display must be considered. A "good" graphics program allows the user to specify the picture or portion of a picture that is displayed [25:53]. However, giving the user too many selections or options can lead to confusion and make the

system difficult to learn [10:284]. The display options must also give the user just the information he desires. Providing irrelevant information not required by the user can crowd the display surface and also lead to confusion.

The analysis of continuous systems being considered in this effort generally uses four graphic tools to examine characteristics of a system: time response, frequency response magnitude, frequency response phase shift, and a root locus. These four items, displayed individually, make up the first four display options. A sample frequency response created using ICECAP-II is shown in Figure IV-2. Other display options are formulated with logical combinations of these four basic options.

The next option implemented consists of the frequency response magnitude and frequency response phase shift displayed simultaneously. Although the phase and magnitude are sometimes placed on the same axes (a Bode plot), they are placed on separate axes in the implementation of ICECAP-II. The following factors influence the decision to use separate axes:

(1.) The dual vertical scale on a Bode plot frequently leads to confusion and misinterpretation especially for inexperienced users.

(2.) Many displays provide only a single linestyle

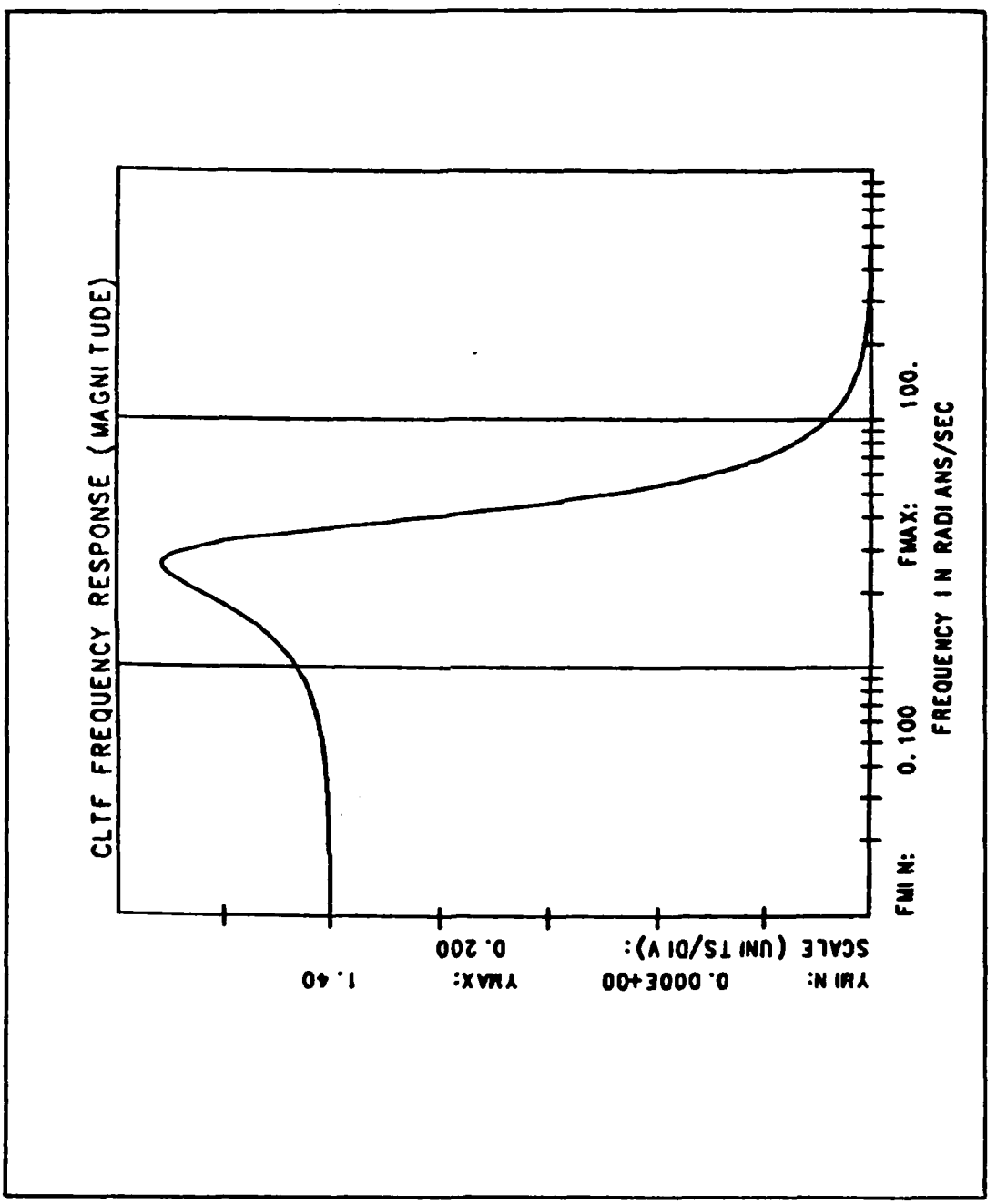


Figure IV-2. Frequency Response Using ICECAP-II

making it sometimes difficult to distinguish between the line representing the magnitude and the line representing the phase. (Using different linestyles such as a solid line for magnitude and a dashed line for phase would improve the situation. Color displays could be used to plot the phase and magnitude in different colors.)

(3.) By plotting the phase and magnitude individually it is simpler to display the single response.

The last option implemented consists of all four items displayed simultaneously. This option allows the user to get an immediate view of all the system characteristics. If necessary, the user can then return to the menu and select a specific response to display for more detailed study.

Thus, ICECAP-II has six display options for the four basic system analysis tools. In each option, a set of FORTRAN logical variables is used to determine the viewport for each plot thus controlling the size and location. This simple manipulation of the viewport allows the same plotting routines to be used regardless of the size and location. One other logical option is the simultaneous display of the root locus and the frequency response magnitude but this is not implemented. Other options can be envisioned but are not as fundamental to system analysis as those discussed

above. The additional commands required for extra options could easily hinder a user rather than aid him.

CORE TEXT CONSIDERATIONS

One of the first problems encountered in implementing ICECAP-II involves a trade-off between execution time and Core generated text. The Core system specification defines three attributes for text known as precisions.

String-precision text is the simplest form and makes almost complete use of the display's character generator. Since this is hardware dependent, it is the fastest method for using the Core system to display text. However, the user has little or no control over the character size, orientation, or spacing [36:II-28].

The next level of text, known as character-precision, still makes use of the hardware character generator but generally gives the application programmer slightly more control over orientation and spacing. As an example, character-precision text can frequently be used to "write" text in an upward or downward direction while string-precision text can only be written from left to right. Since character-precision text positions and writes a single character (as compared to an entire string for string-precision text) it is slightly slower than string-precision text [36:II-28].

The final level of text defined by the Core system is stroke precision text which creates each character using a set of vectors. Since the hardware character generator is not used, complete control of size, orientation, and spacing is possible [36:II-28]. Stroke precision text is used exclusively in ICECAP-II for labeling the axes of the various plots. In addition, stroke precision text allows the text to be automatically enlarged or reduced in size depending on the display option selected. It is necessary to use stroke precision text in cases like this when spacing and size is critical. The stroke precision text is also used to provide a variety of fonts. For example, the GWCORE provides block, double-line block, Roman, and italic fonts. Since each character is formed by a set of vectors, the stroke precision text is significantly slower than the other two types.

The obvious slow speed of stroke-precision text is encountered initially with the very first display created by ICECAP-II. This display simply provides the name of the program, copyright information, and the school's address. A copy of this display (created with stroke-precision text) is shown in Figure IV-3. The time required to create this display ranges from 20 to 30 seconds depending on the memory size of the computer and the number of active users. A comparison of speed versus memory size is contained in Table IV-1. This data was gathered with no other interactive

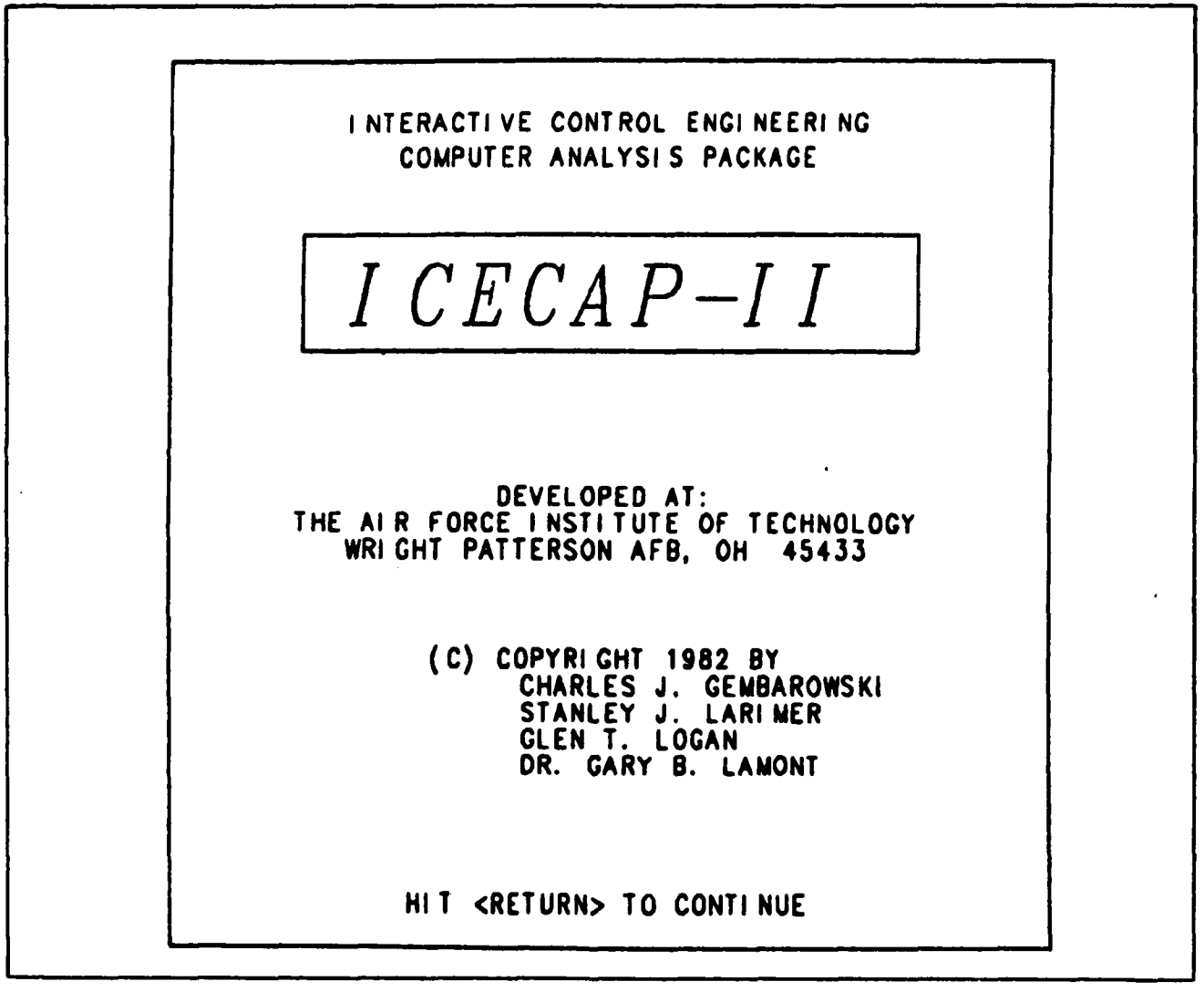


Figure IV-3. ICECAP-II Title Slide Using Stroke Precision Text

users.

MEMORY SIZE	FREE PAGES	ELASPED TIME (sec)	CPU TIME (sec)	PAGE FAULTS
0.50Mb	172	25.13	16.31	440
1.00Mb	775	23.78	15.90	438
2.25Mb	3273	23.66	15.82	424

TABLE IV-1. Execution times for creating Figure IV-3.

The elapsed time increased by as much as three seconds with only two additional users. However, data of this type is difficult to analyze since the time and memory allocations depend heavily on the nature and priority of the jobs run by the other users.

Based on the considerations discussed above, stroke-precision text is used in ICECAP-II only when size and spacing is critical to the information display (such as labeling axes). In all other cases every attempt is made to make full use of string-precision or character-precision text. By using string-precision text for the display shown in Figure IV-3, the elapsed time was reduced to 2.42 sec. (with 1.50 Mb memory and no other users).

USING THE CORE DISPLAY FILE

Contained within the GWCORE is an array of 30,000 integers known as the pseudo display file (PDF). This file

contains all of the vital information about the picture currently being displayed. Typical data found in the PDF includes integer values representing linestyle, font, spacing, color, and strings of x, y, and z coordinates. The coordinates, which are normally real values, are "converted" to integers with a FORTRAN "EQUIVALENCE" statement before being placed in the PDF [43:13]. The PDF is very similar in concept to the GSPC metafile [36:IV]. However, George Washington University did not implement the PDF following the proposed metafile specification.

The primary purpose of the PDF or metafile is to provide a means for storing and transferring graphic pictures. Since the PDF contains all necessary information about a particular picture, only the file must be saved. Several applications for the PDF have been developed at AFIT. One such application reads the PDF, performs several lengthy computations, and finally drives a dot matrix line printer providing the lab with a hard copy capability [32]. Kevin Rose developed another application which he called a metafile translator [29:B-100]. This program allows the user to select an output device and a picture file to be displayed. Displays created on one device can easily be viewed on another using this program. These same capabilities are desireable for ICECAP-II especially for obtaining hard copies. The metafile translator also has value since it permits a user to quickly recall a

particular display without even running the system analysis program.

The necessary code for saving the PDF is included in ICECAP-II but difficulties are encountered in some situations. The problem arises from the length of the PDF and the array space required to store the two display options having multiple plots. Each single plot requires approximately 16,000 to 20,000 of the 30,000 available elements in the PDF. Obviously, putting two or more plots on the same display exceeds the maximum size of the PDF so a complete PDF cannot be obtained in these two instances. In this case, the majority of space in the PDF is used for storing data related to stroke precision text. The x, y, and z coordinates for the start point and end points of each vector used in drawing a character must be stored in the PDF. A simple example of the PDF size is shown in Figure IV-4. This figure compares the contents of the PDF for various precisions and fonts provided by the GWCORE. The display shown in Figure IV-3 consumes 24,198 of the 30,000 elements when stroke-precision text is used. In comparison, the same figure with string-precision text consumes only 1,867 elements of the PDF.

In addition, once the PDF has been filled, program execution slows to nearly a stand-still. The cause of this is the writing of an error message each time an attempt is made to add a new vector to the PDF.

TEXT SAMPLE	PRECISION	LENGTH OF PDF
AFIT GRAPHICS	STRING	61
AFIT GRAPHICS	STROKE	1238
AFIT GRAPHICS	STROKE	2393
AFIT GRAPHICS	STROKE	2406
AFIT GRAPHICS	STROKE	2519

Figure IV-4. Length of PDF for Various Text Styles

In order that all display options could be coded and tested a quick fix to the above problem was devised and implemented. In simple terms, this fix consists of clearing the PDF after drawing each plot (when more than one is to be displayed). This "fools" the GWCORE into believing a fresh PDF is available and allows the program to proceed without any reduction in execution speed. The disadvantages of this approach are:

(1.) The PDF cannot be saved for use with the line printer or metafile translator.

(2.) Selective deletion of segments previously cleared from the PDF cannot be accomplished.

Based on these disadvantages, a long term fix is planned which will enlarge the PDF using an adjustable array or disk file. This change requires several modifications to the GWCORE source code and is not covered in this effort. The quick fix is considered suitable for this effort since it permits demonstration of the display options which is one of the major objectives.

ARRAY PROCESSOR APPLICATIONS

Over the past few years, rapid advancement in computer

architecture has brought about a variety of special purpose computers. One such development is the array processor which is normally installed as a peripheral to a much larger host system. The array processor is designed specifically to perform repetitive operations on large arrays. By utilizing parallel processing techniques, the array processor can frequently perform much faster than the host system thus making the host CPU available for other jobs.

During the implementation phase of this effort, the Digital Engineering Lab acquired a MAP-400 (Macro-arithmetic Array Processor) manufactured by CSPI Inc. The MAP-400 architecture consists of four processors and three buses. Each bus has associated with it a maximum of 64K of RAM with a portion of the first bus memory used for storage of the primary executive. Other than the high speed array processor, there are processors which handle I/O, host interface, and general system control. In addition to the three large memory banks, the MAP-400 provides integer and scalar tables which contain a number of pre-defined and user defined values. With the opportunity to possibly reduce ICECAP-II's execution time, a brief examination of the array processor was included in this effort. This study involved only the graphics applications and not the system analysis functions.

With execution speed being the primary interest, this topic is discussed first. The graphics applications portion

of ICECAP-II uses arrays of medium length to store the data to be plotted. The arrays range in size from 91 to 901 points. (The actual array size of each root locus trace is not known and is problem dependent, but it is expected that it generally falls in the above range). So that a standard window size can be used throughout the program, all data is scaled to fall in a range of zero to one. Of all the modules developed for this effort, it is this scaling process which can benefit most from use of the array processor. In addition to the scaling, the base 10 logarithm of each array element must be computed when a log scale is used for the plot. A data flow diagram for this scaling procedure is shown in Figure IV-5.

With regard to speed versus array size, "the overhead time to setup an array process is independent of the array size. Therefore, the greater efficiency in execution time is achieved if the array size is large [33:8-3]." Generally, operations on arrays 2K or larger are more efficient than the host while operations on arrays with fewer than 100 elements are less efficient than the host. For array sizes between these two limits, the efficiency generally depends on the complexity of the task.

The two programs developed for this test are based on the data flow diagram shown in Figure IV-5. The first program is strictly VAX-11 FORTRAN. The second program is also FORTRAN but all functions are replaced with the

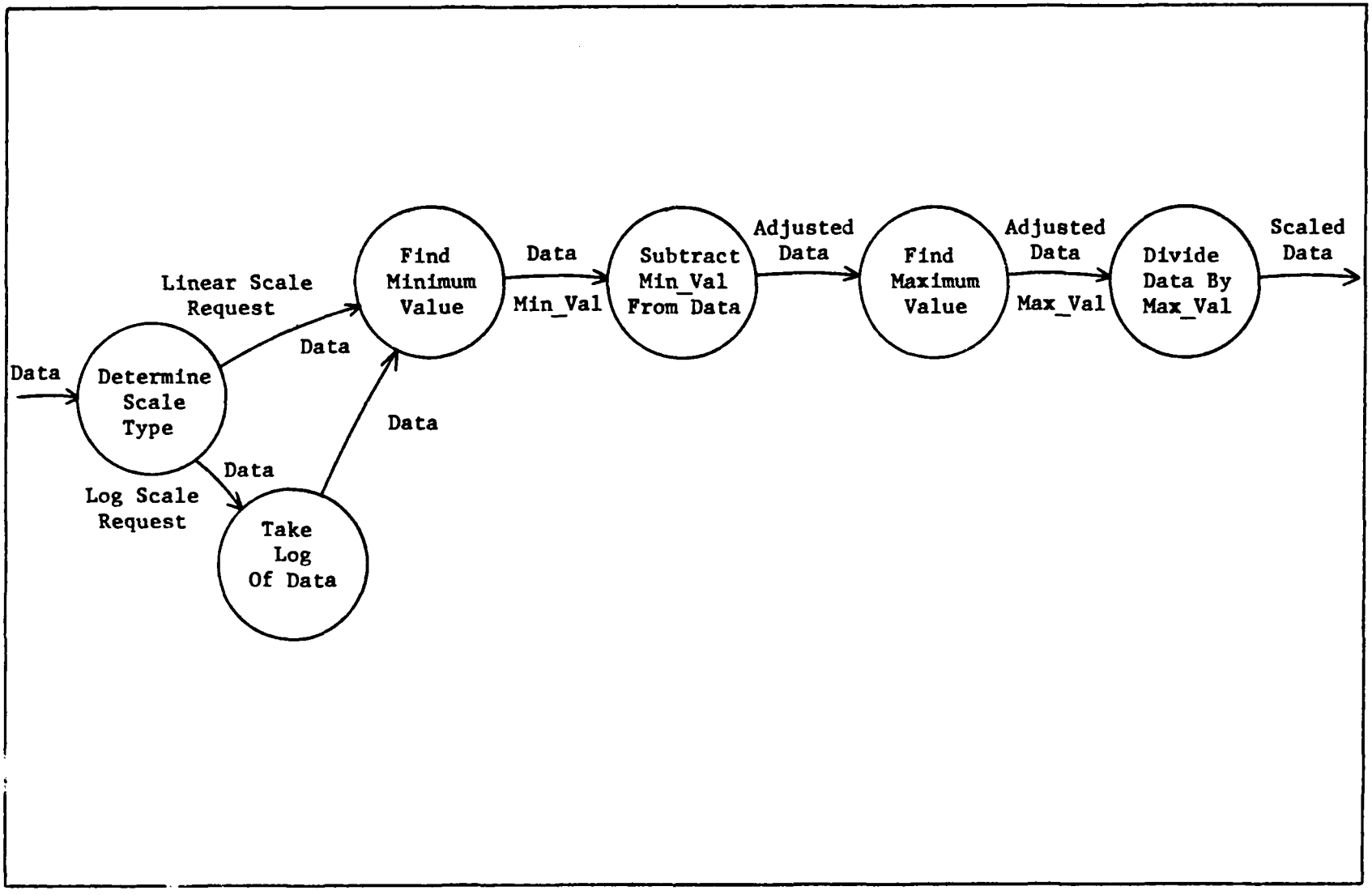


Figure IV-5. Data Flow Diagram for Scaling Routine

equivalent array processor function. In many cases, several lines of FORTRAN code are replaced with a single call to the array processing package known as SNAP-II (Simple Notation for Arrary Processing). In each case the system library functions are used to record elapsed time, CPU time, I/O counts, and page faults. Table IV-2 provides the results for three common array sizes used in ICECAP-II. Each test is performed with 1.50 Mb main memory and no other interactive users.

Based on these results, either the arrays are not long enough or the scaling process is not sufficiently complex to warrant use of the array processor. Only the 901 element array provided a reduction in CPU time (but the elapsed time is still greater). There are programming techniques which can be used to increase program performance while using the array processor. One simple method is to use the host for scalar operations. While the array processor can perform these functions, the parallel processor is not utilized at all so there is essentially no speed improvement available. In fact, the additional time required to transfer scalars from the host to the array processor makes the scalar math very inefficient. The only time scalar math should be performed in the array processor is when the scalar is the result of some array operation. Another technique for improving performance involves pre-binding of several array functions into a new single function that can be invoked by

PARAMETER	VAX-11 FORTRAN	MAP-400
402 DATA POINTS LINEAR AXIS		
ELAPSED TIME (sec)	0.02	0.24
CPU TIME (sec)	0.02	0.09
I/O OPERATIONS	0	39
PAGE FAULTS	0	1
91 DATA POINTS LOG AXIS		
ELAPSED TIME (sec)	0.01	0.21
CPU TIME (sec)	0.01	0.12
I/O OPERATIONS	0	41
PAGE FAULTS	3	1
901 DATA POINTS LOG AXIS		
ELAPSED TIME (sec)	0.14	0.22
CPU TIME (sec)	0.14	0.10
I/O OPERATIONS	0	41
PAGE FAULTS	4	1

Table IV-2. Execution Time for Scaling Process

a single subroutine call. Normally, I/O operations cannot be included in this pre-binding so it is most useful for long complex operations. Pre-binding is not applied to the scaling program discussed above.

Some other important points must be considered when evaluating array processors. First, the SNAP-II system, like many array processing systems, is for only a single user [34:2-85]. So other users must wait for the processor to become available or revert to backup software which runs strictly within the host. Another important consideration is the word format used in the array processor. For example, the MAP-400 performs floating point operations on 16 or 32 bit data [33:2-1]. This does not permit the user to write application programs using double precision or 16 byte real variables which are provided by VAX-11 FORTRAN. The final consideration is accuracy. While the array processor may provide a speed advantage, it is found that the MAP-400 makes a sacrifice in accuracy. A simple test with logarithms shows that the MAP-400 agrees with VAX-11 FORTRAN to only about five decimal places. When evaluating the logarithm of one to twelve decimal places, VAX-II FORTRAN provides the expected result of zero while the MAP-400 provides a result of 1.148338E-6. Inaccuracies such as these can sometimes be catastrophic to application programs.

While the array processor provides no benefit to this

specific project, it may have significant value in both graphics and system analysis. In graphics, the array processor could be employed to perform viewing transformations where large arrays must be manipulated. Rotation of a picture could probably be handled quite efficiently by an array processor. In system analysis, the applications are almost unlimited. Vector multiplications, matrix inversions, and FFTs can be performed with single subroutine calls. The potential applications of the array processor certainly deserve more investigation than the simple application considered in this effort.

SUMMARY

This chapter has discussed the complete implementation of ICECAP-II including several of the problems encountered. Based on the nature of this effort, a combination of top-down and incremental strategies was used for ICECAP-II. This was the most logical strategy since a large amount of software had been coded and tested in previous efforts. The rationale behind each display option was discussed and the use of the viewport to implement these options was presented. Other graphic topics discussed in this chapter include the size of the pseudo display file and the use of Core generated text. Several examples were provided to compare the advantages and disadvantages of the different types of text. Finally, an array processor was examined for

possible use in ICECAP-II to perform scaling functions. Although it did not benefit ICECAP-II in this case, there are significant applications for the array processor in both graphics and system analysis that deserve more investigation.

CHAPTER V

TESTING

INTRODUCTION

Software testing is the process of exercising a program to demonstrate that it performs the desired functions. However, to be of any value, the time spent testing a program should be aimed primarily at finding errors [23:5]. By seeking out and correcting these errors or bugs, the value of the program to the user is increased. This chapter discusses some of the various strategies and methods for software testing. Some typical errors will be discussed as well as the test techniques for finding these bugs. The remainder of the chapter is then oriented to the testing of ICECAP-II which is divided into two phases: graphic performance testing and general functional testing. The test plan and samples of the test results can be found in Appendix C.

DEVELOPING A TESTING STRATEGY

There are a number of techniques available for testing software. Generally, these methods can be grouped into two broad categories: functional or structural testing [3:4-5]. These categories are quite different in their approach and

each has unique advantages and disadvantages. Neither method, however, provides a complete test and there exists no foolproof method that does. As will be seen in the following paragraphs, complete testing "is not theoretically possible, and certainly not practically possible [3:12]."

Functional testing exercises a program from a "black box" point of view to determine if the program satisfies the specified functional requirements. This type of testing is really concerned with the user's point of view since the testing is based on a variety of inputs and the corresponding outputs [3:4]. Although it may be possible to test each function, the variety of user inputs is essentially unlimited thus supporting the idea that complete testing is not possible.

Structural testing is more concerned with implementation details such as programming style, control methods, and data-base design [3:5]. One goal of structural testing is to exercise each possible path through a program at least once. For a large program this task is mind-boggling and "even the number of paths through a small routine can be awesome, because each loop multiplies the path count by the number of times through the loop. A small routine can have millions or billions of paths [3:13]." It is easy to see why structural testing is also incomplete. For example, just determining the number of paths through a program is a difficult task. Beizer provides a maximum path

arithmetic for estimating the quantity of paths and determining the probability of executing a particular path [3:122-133]. With this information in hand, Beizer contends that structural testing be heavily "biased toward the low probability paths rather than the high probability paths [3:129]."

Myers also discusses the same two general categories but calls them by a different name. Black box testing is the name applied to the functional test category and white box testing is used to describe the structural test category [42:8-10]. Myers draws the same conclusions as Beizer for each category:

> (1.) exhaustive input testing for black box tests is not practical or possible [23:8].
> (2.) exhaustive path testing for white box tests is neither practical nor possible.

Myers also emphasizes an important disadvantage of white box testing. Simply causing every program statement to execute at least once is not sufficient to conclude that a program is error free. A simple example helps to illustrate this point. If a program is written to add two numbers but mistakenly multiplies them, an exhaustive path test is of no value at all. Such a bug as this can only be

discovered through black box or functional tests. Generally, "an exhaustive path test in no way guarantees that a program matches its specification [23:10]."

Since heavy emphasis is placed on the functional requirements of ICECAP-II, it is tested only from the black box or functional viewpoint. Also, since the main objective of ICECAP-II is to demonstrate the feasability of using interactive graphics for system analysis, functional testing is the best method available to show that the objective is satisfied. Finally, the detailed software analysis and development of a path testing scheme necessary for white box testing is far too complex for this effort.

Since exhaustive testing is not possible, it is desirable to test those areas which commonly contain bugs or errors. Functional testing can be divided into many smaller categories in a tree-like fashion but perhaps the most common bugs, in Beizer's terminology, are the process errors. This category is composed of arithmetic, initialization, control, and logic types of errors [3:23-25]. Arithmetic errors generally occur when coding algebraic expressions. Programming languages generally require special symbols and strict attention to operator precedence and simple oversights can have unexplainable results. The other process errors have similar results and Myers adds to the list of common bugs with subscript or indexing errors, division by zero, and mixed mode

comparisons [23:30]. Because these types of errors are most common they are the primary suspect when any functional test fails.

A knowledge of error types is important in the development of test cases so that the testing can be aimed at finding these errors. The testing of ICECAP-II is divided into two phases: graphic performance testing and general functional testing. The test cases for each phase are discussed under the corresponding sub-heading.

GRAPHIC PERFORMANCE TESTING

The primary purpose of the graphic performance testing is to find and correct errors in the graphics portion of ICECAP-II which constitutes the majority of the coding for this effort. This is actually a subset of the functional testing but since the test method is quite different, it is discussed seperately.

Although the user does not make any direct input to the graphic routines, these routines are responsible for a significant amount of output. The basic test approach in this phase is to compare the graphics output with the array of data points. These are considered very valuable tests for if the graphic routines do not accurately plot the data points, then ICECAP-II is virtually useless. Since the number of data points is quite large, it is not practical to

verify each point. It is easiest to select points that are easily obtained from the plot. These include minimum values, maximun values, zero crossings, and intersections with marked grid lines. A sample result for each of the four basic plots is included in Appendix C. It is not necessary to test the display options for accuracy since the same routines are used to do the plotting but in a different viewport. In any case which the plotting routine does not accurately represent the data, the source code will be examined for bugs starting with a search for one of the process errors. It should be noted that evaluating the graphical accuracy is a partially subjective test. Both hardware and software limitations can affect the accuracy of plotting a number containing seven or eight decimal places. Thus, if a data point is not accurately graphed, the tester must make a subjective judgement of whether this is a system limitation or a programming bug. Generally, a small number of minor deviations is caused by system limitations while numerous differences of two to three percent or more warrants a search for program bugs. ICECAP-II successfully passed this test phase. Samples of the results are in Appendix C.

GENERAL FUNCTIONAL TESTING

The remainder of the ICECAP-II testing is the standard functional or black box tests which involve no subjective

evaluations. This is the strict input-output testing where various inputs are made to the system and the outputs are compared to the expected or desired result. The first part of this test verifies that each valid input obtains the correct or desired output. This shows that ICECAP-II satisfies its basic objective. This type of test is also used at each incremental implementation step (each time a module is added or modified). When used in this way, it is sometimes referred to as incremental testing. However, as both Beizer [3] and Myers [23] recommend, the majority of testing is aimed at finding errors caused by invalid input. Beizer outlines five test cases that are likely to find program errors and these are used in testing ICECAP-II [3:157].

1. "Do it wrong." Make an erroneous or invalid input. If the program asks for a number from one to five, enter six. (This is sometimes called a boundary value test.)

2. "Wrong combination." Enter an invalid com[illegible] commands and/or data.

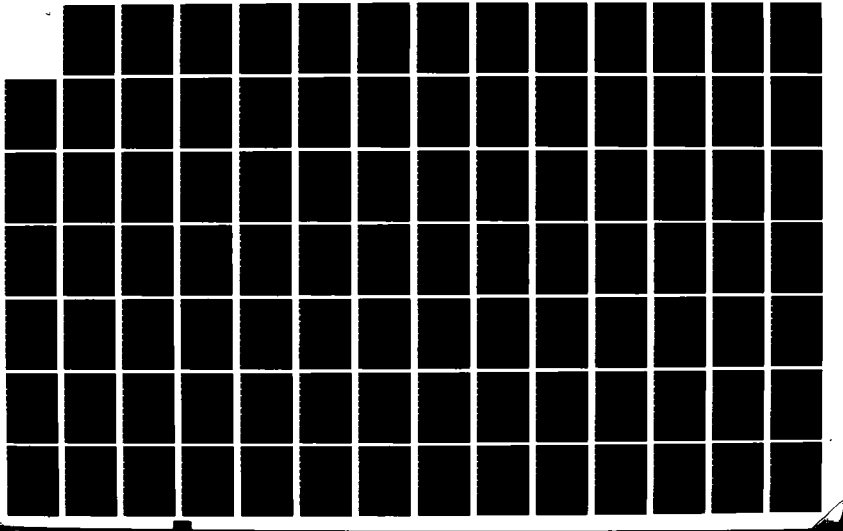
3. "Don't do enough." If the progra[illegible] values, enter one or two. Enter an in[illegible]

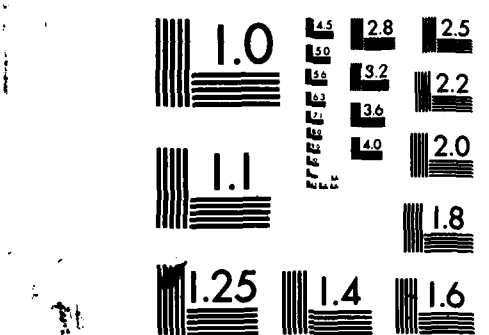
4. "Don't do nothing." Simply h[illegible] of-command signal.

5. "Do too much." Ent[illegible] provide more data than re[illegible]

Any errors re[illegible] obvious and the[illegible] cases, add[illegible]

AD-A138 025 INTERACTIVE COMPUTER GRAPHICS FOR SYSTEM ANALYSIS(U) 2/3
AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOOL
OF ENGINEERING M A TRAVIS DEC 83 AFIT/GE/EE/83D-66
UNCLASSIFIED F/G 5/1 NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

erroneous input but the extra protection it provides to the user makes it worthwhile. Various samples are included in Appendix C.

Beizer refers to one other type of functional check which he calls ad-lib testing. The ad-lib test is the most informal of all testing since the tester is usually a casual bystander with little or no knowledge of the system. In many cases, the bystander is just curious and wishes to execute sample problems but at the other extreme is a "hotshot operator who's earned a reputation for crashing any system in under two minutes... [3:168]" In either case, there is no documented test plan and 90% to 95% of the ad-lib tests consist of input strings with format violations [3:168]. If the black box testing is as comprehensive as possible then the ad-lib testing is not likely to uncover any new bugs. However, ad-lib testing is also useful for collecting general comments and user reactions to the system so it is used in this effort to supplement the testing discussed in the preceding paragraphs. ICECAP-II completed these tests with only one major bug being discovered. Sample results can be found in Appendix C.

SUMMARY

Software testing aimed at finding errors can provide improvements to a program which are beneficial to the users. Either black box (functional) or white box (structural)

testing can be used, but based on the objective of this effort and the complexity of path testing, a functional approach is used to test ICECAP-II. Graphic tests and standard input/output tests are used to find problem areas which, when corrected, improve both the program and the user interface. Finally, ad-lib testing is used to further test input and output and collect general comments on the user interface.

CHAPTER VI

CONCLUSION & RECOMMENDATIONS

The successful design and implementation of ICECAP-II demonstrates the real value of interactive graphics for system analysis. The high resolution available on modern displays provides output to the user that is more pleasant to view and easier to interpret than the alpha-numeric character plots still in use on many systems. Also, the extensive capabilities of the Core system allow for simple implementation of a variety of display options. Such options permit the user to display only the information he desires. These options with the high resolution graphics provide a significant improvement to the user interface that has not been available in the past. Probably most important is the fact that AFIT's Digital Engineering Lab now has a functional tool for continuous systems analysis and that tool has all of the advantages discussed above.

The success of this effort has also proven the validity of the design method used. The basic structured design process used for ICECAP-II is useful for developing an entirely new system or adding new capabilities to an old one as in this effort. In either case it is most important to begin with a complete specification of the functional

requirements. Once the functional requirements are established and fully understood, the design process continues with the development of the user's model or mental picture and a hierarchy of data flow diagrams. The command language is also an important consideration at this stage of design and several methods are discussed in this effort. A combination of menus and keyboard commands, however, are selected for ICECAP-II. This has proven to be an effective method primarily because of the flexibility available for entering commands. An experienced user can enter a complete command without viewing any menus while a novice user or one who cannot recall a particular command can traverse a tree-like structure of menus until the desired command is formulated. This method was originally used by Gembarowski for the original ICECAP. It was simply expanded for ICECAP-II to accomodate the additional capabilities required by the functional specification and the user's model. It is also noted that providing the user with more commands than absolutely necessary is not necessarily helpful. Too many commands clutter the menus and only confuse the user. For this reason only six display options are implemented for the four basic plots. The remainder of the structured design method used in ICECAP-II is straight forward and consists of structure charts and data dictionaries.

With a solid design, the implementation proceeds quickly and relatively free of bugs. A hybrid of top-down

and incremental implementation approaches further simplifies the coding and debugging process. This method is very useful, especially when modifying or making additions to existing software.

Finally, ICECAP-II was successfully tested using a functional or input/output approach. Since ICECAP-II is intended to demonstrate a user oriented system, the black box approach is ideal since it tests primarily from the user's point of view.

The above paragraphs contain a brief summary of this entire effort with an attempt to mention the most important considerations and conclusions. It is felt that many of these points are applicable, in a general sense, to any graphics application for system analysis. These methods are by no means perfect, complete, or optimum but they do provide invaluable experience and a good starting point for additional work. A number of possible research areas come to mind as a result of this effort and are discussed in the following section.

RECOMMENDATIONS

(1.) The most obvious area deserving further attention is the expansion of ICECAP-II to include discrete time system analysis and stochastic analysis. These are the priority two and three requirements discussed at the end of

Chapter II. Since this represents a significant amount of work, it could probably best be accomplished with two individual efforts: one for the priority two functions and another for priority three. Even if the stochastic analysis functions are developed at a later time, ICECAP-II could be placed in general use following successful testing of the discrete time functions. There is significant value in computer graphics for system analysis and it should be introduced into the classrooms and labs as soon as possible.

(2.) A significant problem encountered throughout this effort involves overflow, underflow, and invalid arguments to math functions. These problems generally occurred within the VAX TOTAL portion of ICECAP-II and since there is no type of protection the program aborts and provides a cryptic error message. This can be extremely frustrating to any user especially if significant results or data are lost when the program is aborted. Simple protection could be implemented in one of the efforts discussed above but a more detailed investigation could develop general purpose methods for detecting or predicting overflow (and underflow), testing for invalid math arguments, and recovering from such errors. Such general purpose methods would be of value not only to ICECAP-II but to any application program.

(3.) Even though a simple application for an array processor was considered in this effort, there exists a great number of opportunities for its use in system

analysis. A detailed study could identify all of the possible applications and conduct various performance evaluations. This study could help other application programmers determine when and where to employ array processors in their work. There may be available a significant reduction in execution time for some programs. Other important factors such as machine dependencies, freeing the host CPU for other jobs, and accuracy must also be considered.

(4.) During this effort a simple demonstration of color graphics was conducted. The visual impact of a color display is very exciting and there is a variety of possible uses. For example, some parameter of a control system could be varied after the initial analysis and the new responses could be plotted, in a different color, over the previous responses. A study of color graphics for system analysis could identify and examine all of the various possibilities. However, the resolution of the color display currently in use is not really sufficient for data plotting so it is also recommended that the planned acquisition of high resolution color terminals be expedited.

(5.) Finally, since ANSI has elected to adopt GKS as the standard for graphics, a detailed study is in order to compare the GKS with Core. As discussed earlier in this report, the GKS has no relative primitives. The relative lines and movements provided by the Core were found to be

quite useful in ICECAP-II but a more serious comparison is necessary. The study should also recommend which standard be used by programmers in future graphic applications. Use of both the Core and the GKS could result in wasteful duplication and serious imcompatibilities.

In closing, the development of ICECAP-II has been an interesting and challenging effort. Further efforts in computer graphics for system analysis will provide the students and designers with a valuable tool that has not been available in the past.

BIBLIOGRAPHY

1. Astrom, K.J. and H. Elmquist, "Perspective on Interactive Software for Computer Aided Modeling and Design of Control Systems", Proceedings of the 20th IEEE Conference on Decision and Control, 1981.

2. ANSI/X3 News Release, Oct 8, 1982 "ANSI Begins Steps to Adopt GKS"

3. Beizer, Boris. Software Testing Techniques. New York: Van Nostrand Reinhold Co., 1983.

4. Booth, Kellog S. Tutorial: Computer Graphics, New York: Institute of Electrical and Electronics Engineers, 1979.

5. Brown, Marlene "Graphics Standards Advancing on Many Fronts" Software News (3), 6; June 1983.

6. Curling, Harold. "Design of an Interactive Input Graphics System Based on the ACM Core Standard", Master's Thesis, Air Force Institute of Technology, Dec 1980.

7. D'Azzo, John J. and Constantine H. Houpis. Linear Control System Analysis and Design, 2nd ed. New York: McGraw Hill, 1981.

8. Foley, James D. and Victor L. Wallace, "The Art of Natural Graphic Man-Machine Conversation", Tutorial: Computer Graphics, New York: IEEE. 1979.

9. Gembarowski, Charles J. "Development of an Interactive Control Engineering Computer Analysis Package for Discrete and Continuous Systems," Master's Thesis, Air Force Institute of Technology, 1982.

10. Green, Thomas, "Programming as a Cognitive Activity." in Human Interaction With Computers ed. H.T. Smith and T.R.G. Green, pp 271-320. London: Academic Press, 1980.

11. Gudmundsson, Bjorn "User Level Concepts in an Interactive Picture Processing System", Proceedings of 5th International Conference on Pattern Recognition, Los Alamitos, CA: IEEE Computer Society Press, 1981.

12. IMSL Library Reference Manual, 9th ed. Houston: IMSL Inc., 1982.

13. Jensen, Randall W. and Charles C. Tonies. Software Engineering. Englewood Cliffs, NJ: Prentice Hall, 1979.

14. Jones P.F., "Four Principles of Man Computer Dialogue", Tutorial: Computer Graphics, New York: IEEE, 1979.

15. Keller, Pete, et al. GRAFLIB Reference Manual. Livermore, CA: Lawrence Livermore Laboratory, 1980.

16. Langhorst, Fred E. "Working Toward Standards in Graphics" Computer Design (21), 7; July 1982.

17. Larimer, Stanley J. "An Interactive Computer Aided Design Program for Digital and Continuous Control System Analysis and Synthesis" Master's Thesis, Air Force Institute of Technology, Mar 1978.

18. Larimer, Stanley J. TOTAL Users Manual (CAD), Wright Patterson Air Force Base, Ohio: Air Force Institute of Technology, June 1981.

19. Logan, Glen T. "Development of an Interactive Computer Aided Design Program for Digital and Continuous Control System Analysis and Synthesis," Master's Thesis, Air Force Institute of Technology, 1982.

20. McGillem, Clare D. and George R. Cooper. Continuous and Discrete Signal and System Analysis. New York: Holt, Rinehart, and Winston, 1974.

21. Moore M.V. and L.H. Nawrocki, The Educational Effectiveness of Graphic Displays for Computer Aided Instruction. Technical Paper 332. Alexandria, Virginia: US Army Research Institute, September 1978.

22. Moynihan, John A. "What Users Want", Datamation, April 1982.

23. Myers, Glenford J. The Art of Software Testing. New York: John Wiley & Sons, 1979.

24. Myers, Ware. "Computer Graphics: The Human Interface", Computer, June, 1980.

25. Newman, William M. and Robert F. Sproul. Principles of Interactive Computer Graphics, 2nd ed. New York: McGraw Hill, 1979.

26. Organick, Elliot I, Alexandra I. Forsythe, and Robert P. Plummer. Programming Language Structures. New York: Academic Press, 1978.

27. Peters, Lawrence J., Software Design: Methods & Techniques, New York: Yourdon Press, 1981.

28. PLOT-10 User's Manual. Beaverton, OR: Tektronix, Inc., 1971.

29. Rose, Kevin W. "Development of an Interactive Computer Graphics System Library and Graphics Tools," Master's Thesis, Air Force Institute of Technology, 1982.

30. Shneiderman, Ben "Human Factors Issues in Designing Interactive Systems" (abstract). Proceedings of COMCON Fall 1981. IEEE Computer Society Press: Los Angeles, 1981.

31. Shneiderman, Ben. "How to Design with the User in Mind", Datamation, April 1982.

32. Simmons, Matt. PRNTRNX Extension to the Core Graphics System. Unpublished. Air Force Institute of Technology, 1982.

33. Simple Notation for Array Processing (SNAP-II) Volume I. Billerica, MA: CSPI Inc., 1981.

34. Simple Notation for Array Processing (SNAP-II) Volume II. Billerica, MA: CSPI Inc., 1981.

35. Sonderegger, Elaine L. "Recent ANSI X3H3 (Computer Graphics) Activity" Computer Graphics (16), 4; Dec 1982.

36. "Status Report of the Graphics Standards Planning Committee ACM/SIGGRAPH" Computer Graphics (13), 3; August, 1979.

37. Tarbell, Philip B. "Continued Development and Implementation of a Standard Graphics Package for the AFIT VAX 11/780", Master's Thesis, Air Force Institute of Technology, 1981.

38. User Document for the GWU Core System on the VAX 11/780. Washington DC: George Washington University, 1981.

39. Users Manual for the Tektronix 4014 and 4014-1 Display Terminal. Beaverton, OR: Tektronix Inc., 1974.

40. VAX-11 FORTRAN Language Reference Manual. Maynard, MA: Digital Equipment Corp., 1982.

41. VAX-11 PASCAL Primer. Maynard, MA: Digital Equipment Corp., 1980.

42. VAX-11 PASCAL Reference Manual. Maynard, MA: Digital Equipment Corp., 1981.

43. Wenner, Patricia, et al. Design Document for the George Washington University Implementation of the 1979 GSPC Core System. Washington DC: George Washington University, 1980.

44. Yourdon, Edward and Larry L. Constantine. Structured Design, 2nd ed. New York: Yourdon Press, 1978.

APPENDIX A

PROGRAMMER'S MANUAL FOR ICECAP-II

Presented to the Faculty of the School of Engineering

of the Air Force Institute of Technology

Air University

in Partial Fulfillment of the

Requirements for the Degree of

Master of Science

by

Mark A. Travis

CAPT USAF

Graduate Electrical Engineering

December 1983

APPENDIX A

TABLE OF CONTENTS

APPENDIX-A

PROGRAMMERS MANUAL FOR ICECAP-II

INTRODUCTION

This manual is primarily intended for those programmers who will be modifying or increasing the capabilities of ICECAP-II. All of the design documentation for this initial effort is contained in the following pages and includes the data flow diagrams, structure charts, data dictionaries, and file information. Although most of the information is aimed specifically at ICECAP-II, the data flow diagrams are fairly general in nature and could be used as the basis for the development of an entirely new system for any machine and any language. Some other data in this manual, particularly the information on files and linking, is VAX dependent and may have to be modified for use on other machines.

DATA FLOW DIAGRAMS

The data flow diagrams on the following pages represent the initial design of ICECAP-II. These diagrams are the first step taken towards transforming the conceptual requirements into actual software. Each diagram identifies, in a hierarchical fashion, the major processes and data elements which make up the system. Since these diagrams

were developed before deciding to use the large amount of existing software, all of the system analysis functions are shown in addition to the graphics operations. Following the decision to use the existing software, these diagrams proved helpful in identifing the appropriate points to interface the new graphics functions with existing system analysis programs.

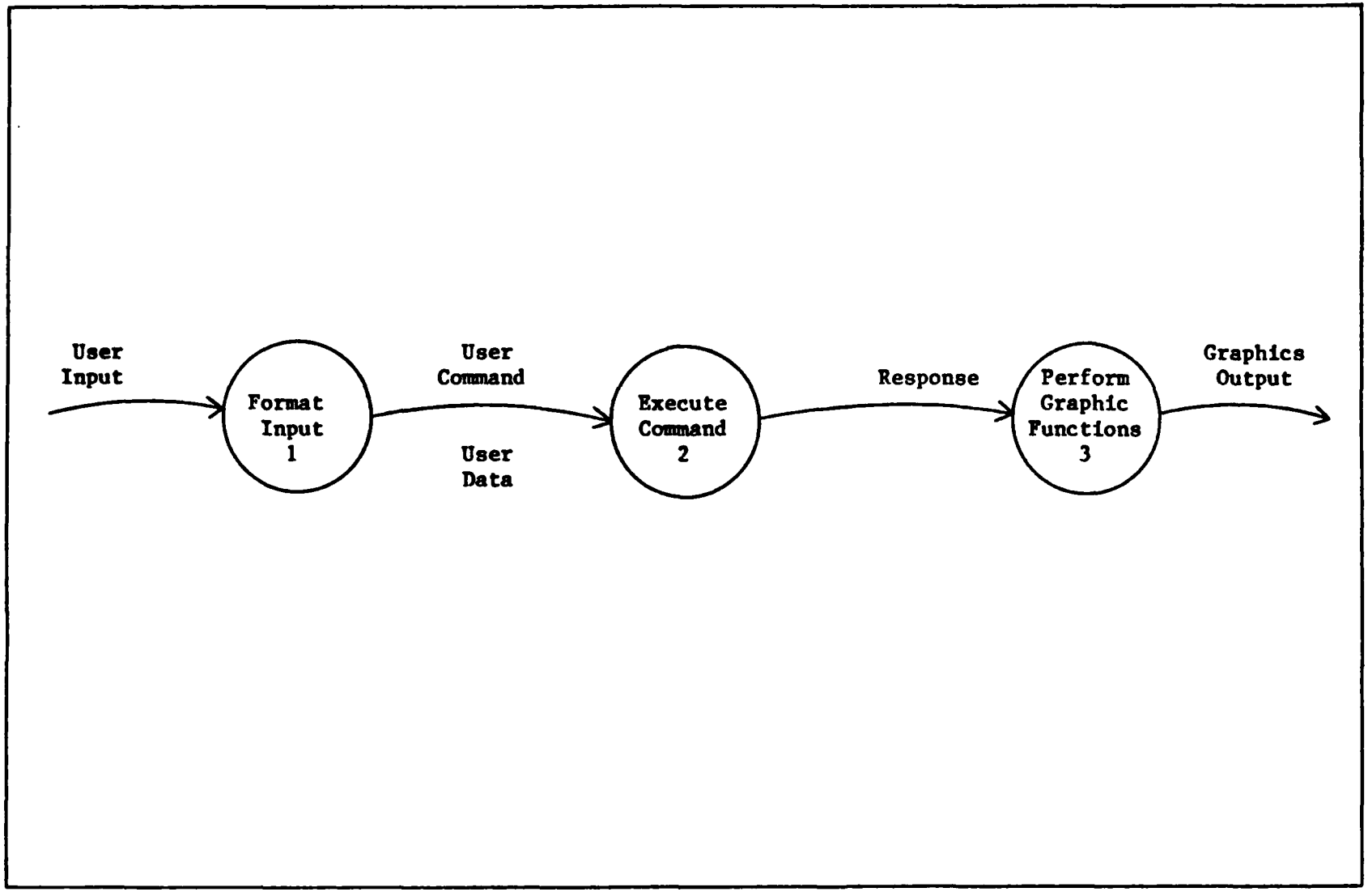


Figure A-1. Overall System Diagram

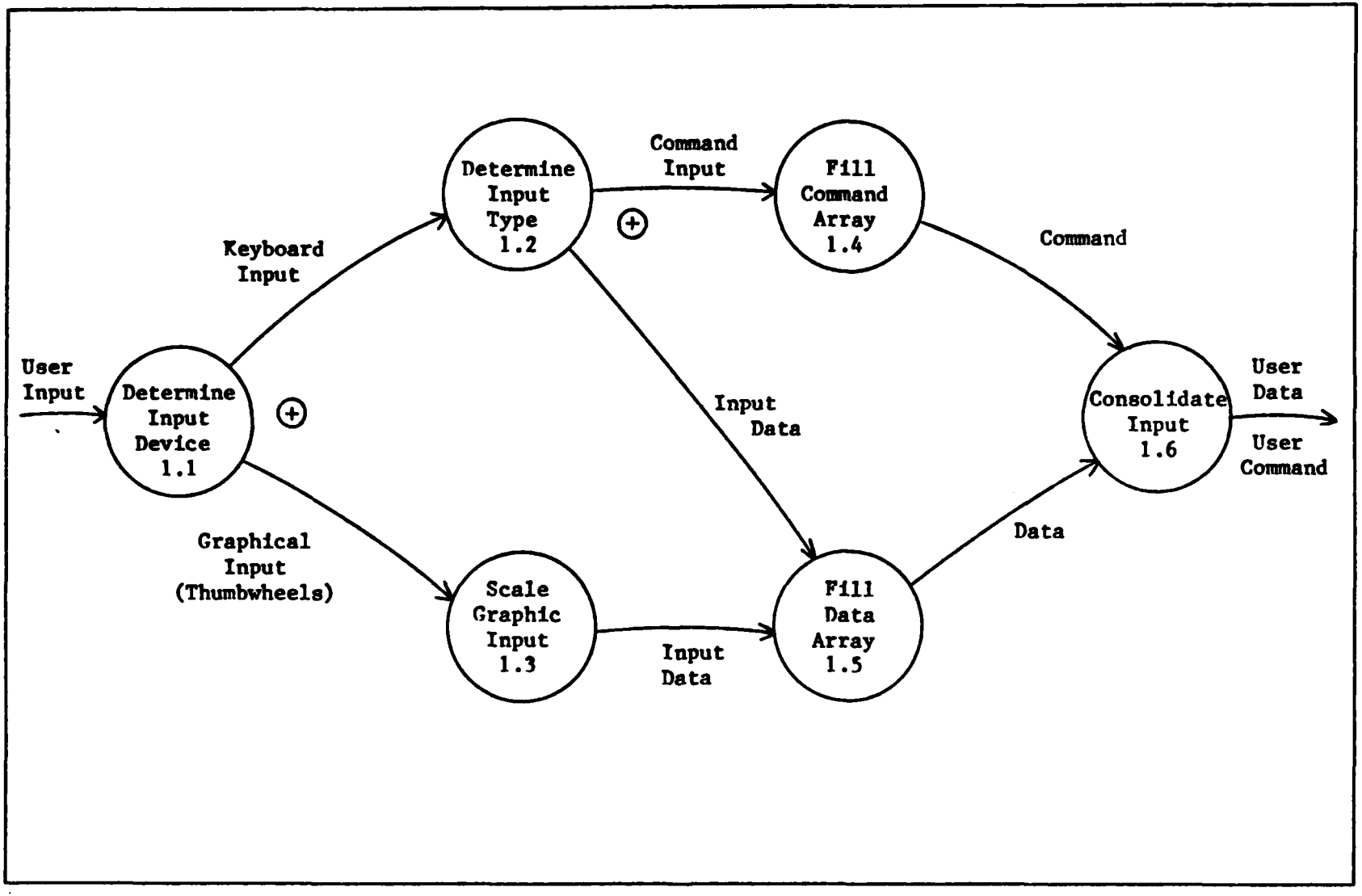


Figure A-2. Format Input (node 1)

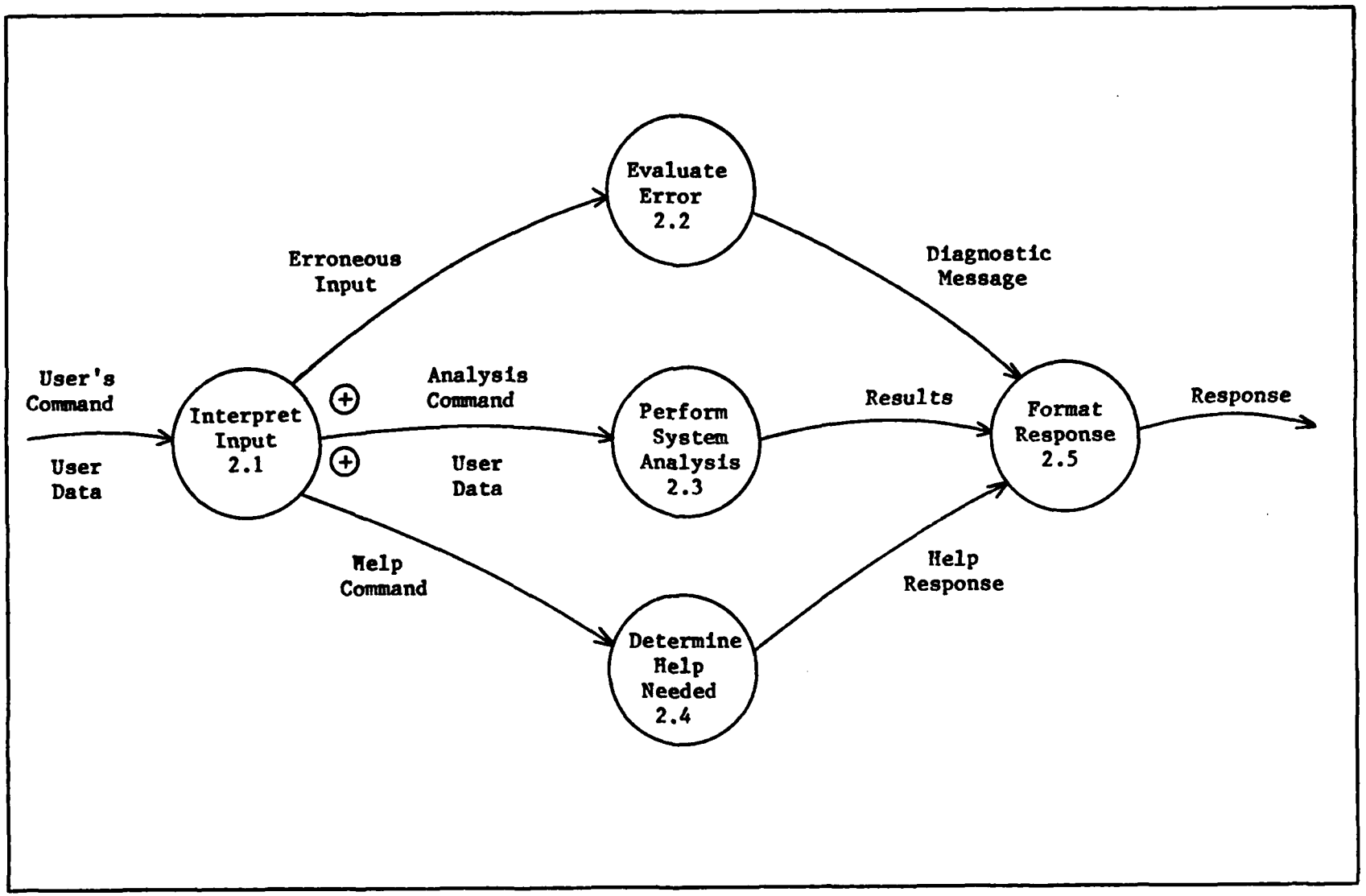


Figure A-3. Execute Command (node 2)

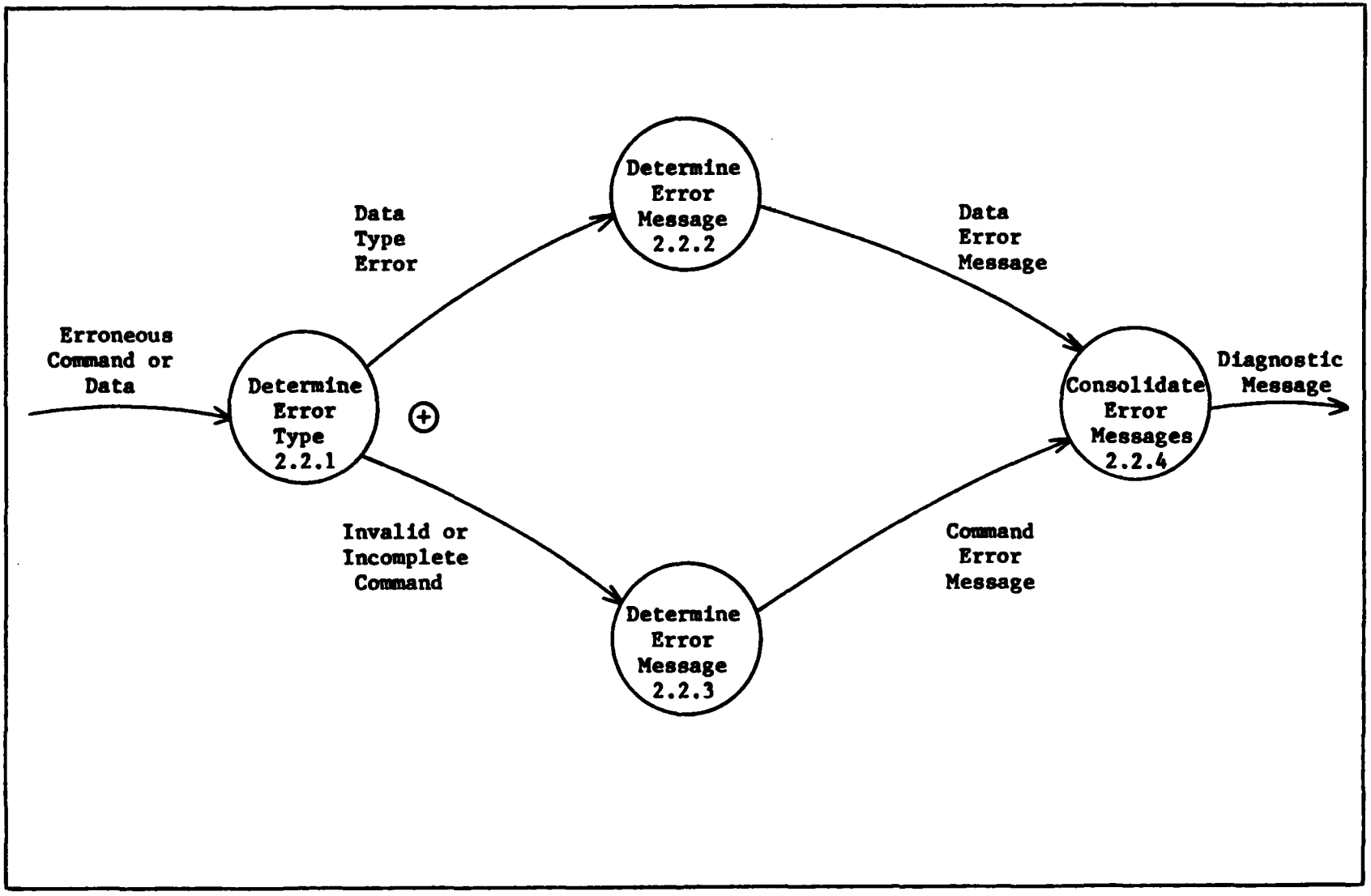


Figure A-4. Evaluate Error (node 2.2)

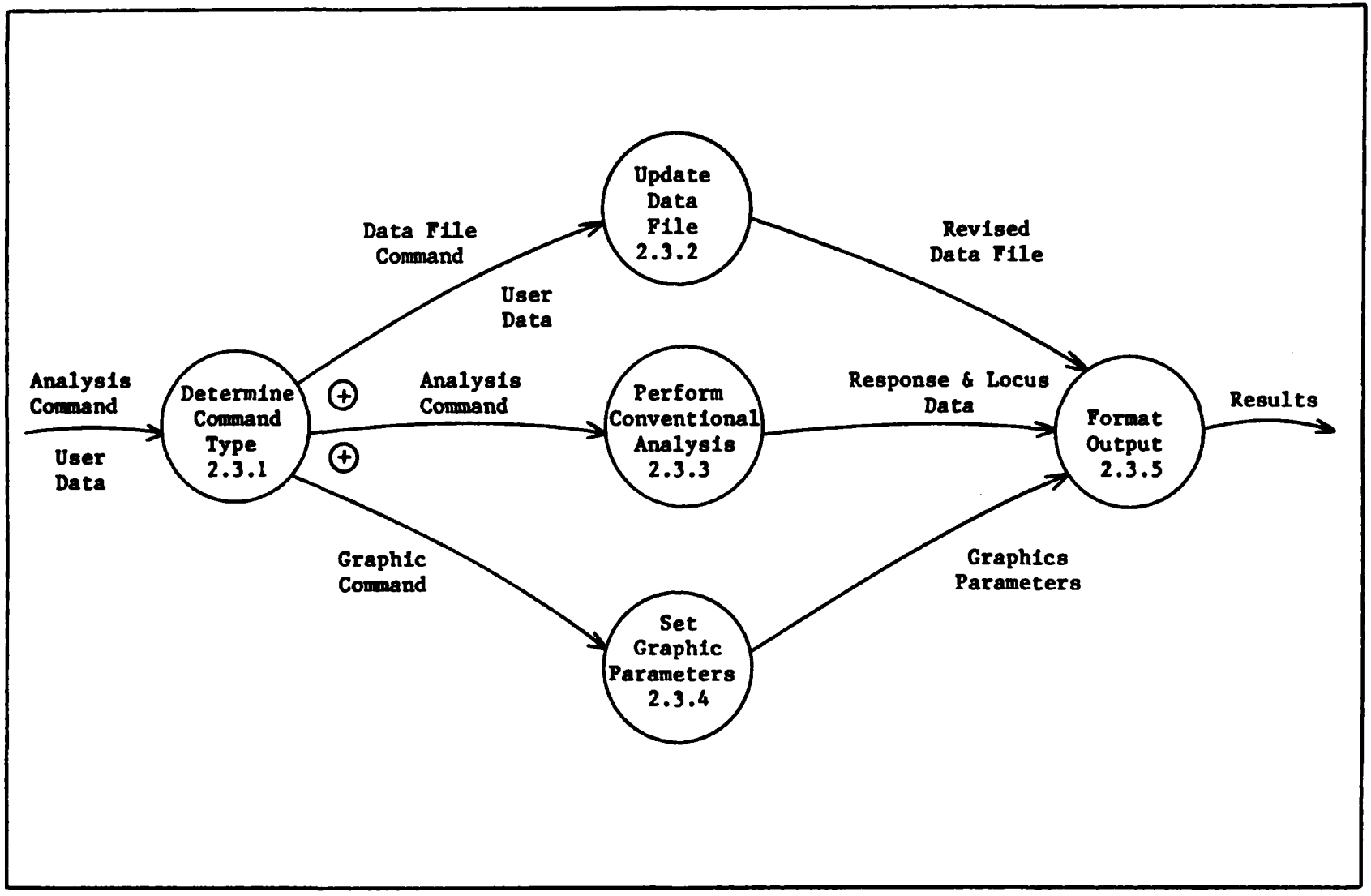


Figure A-5. Perform System Analysis (node 2.3)

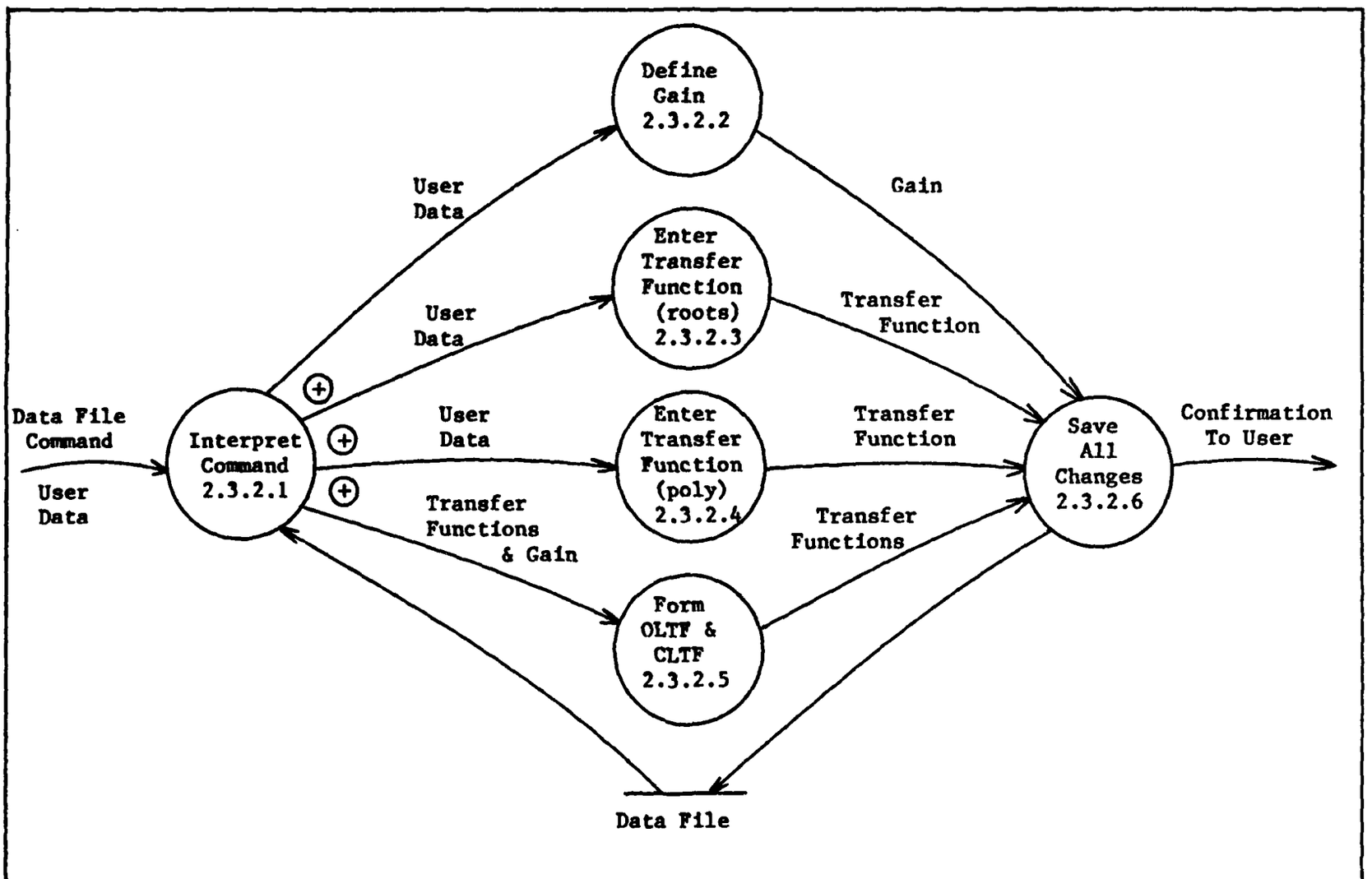


Figure A-6. Update Data File (node 2.3.2)

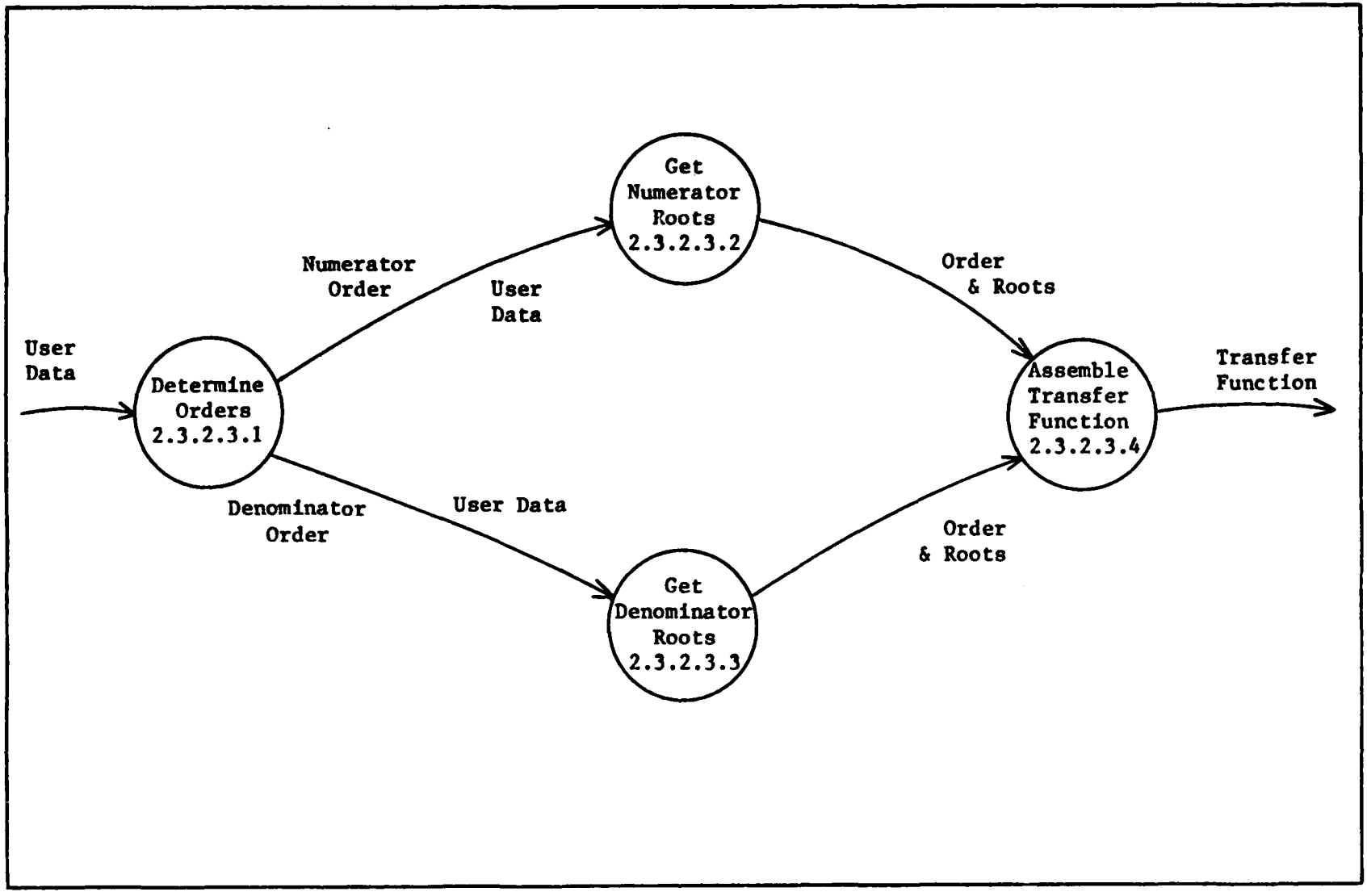


Figure A-7. Enter Transfer Function (roots) (node 2.3.2.3)

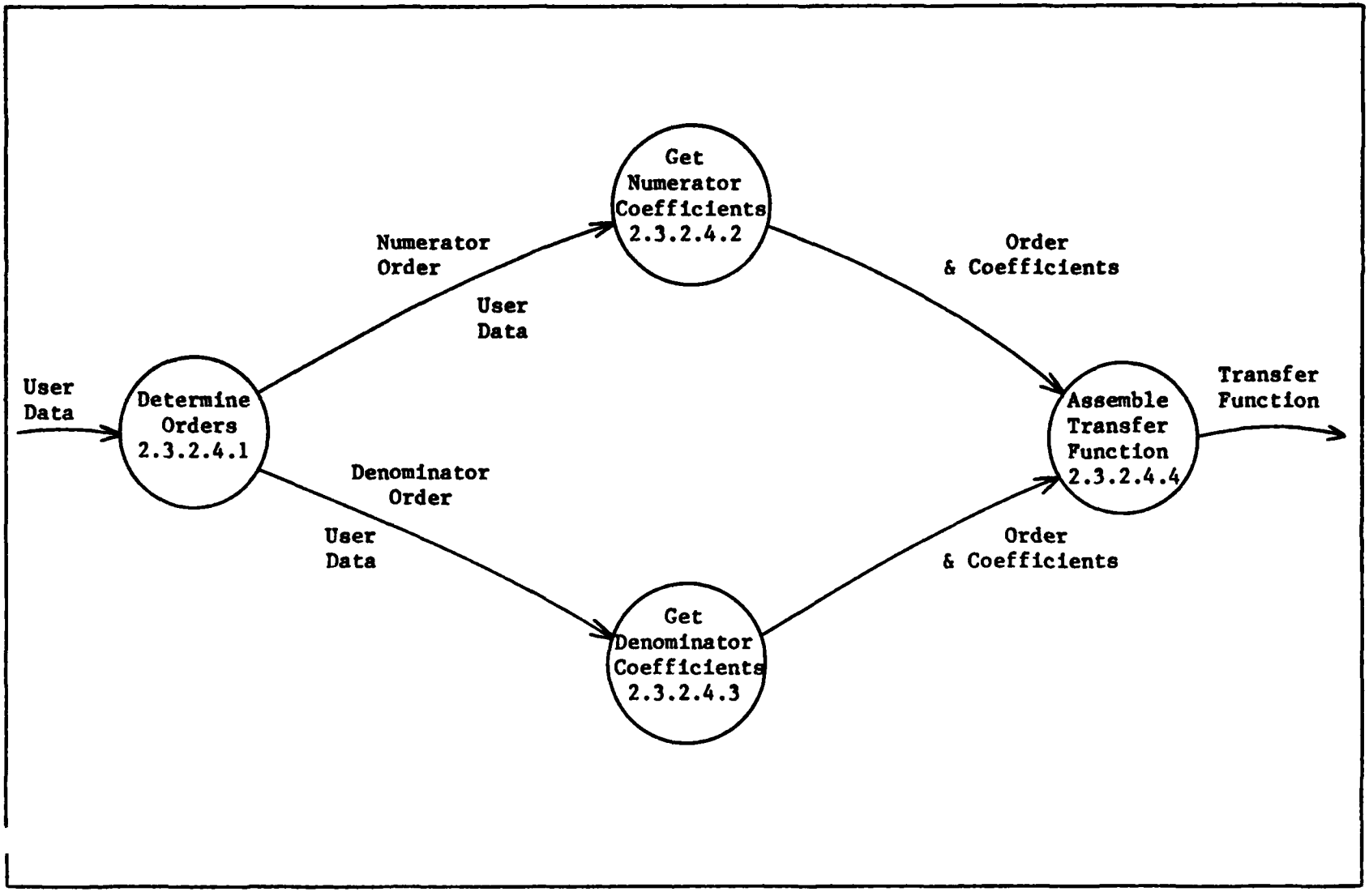


Figure A-8. Enter Transfer Function (polynomial) (node 2.3.2.4)

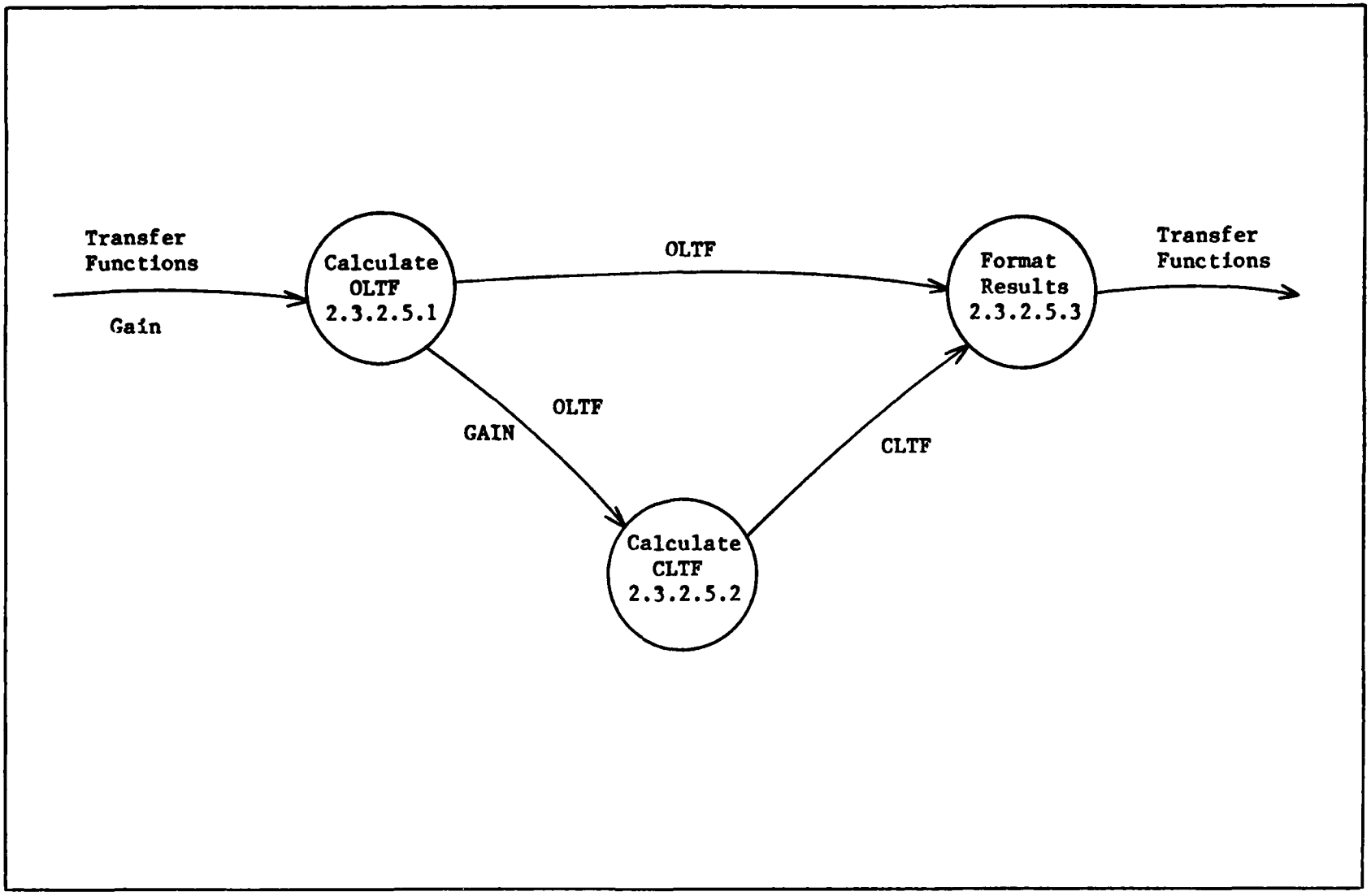


Figure A-9. Form OLTF & CLTF (node 2.3.2.5)

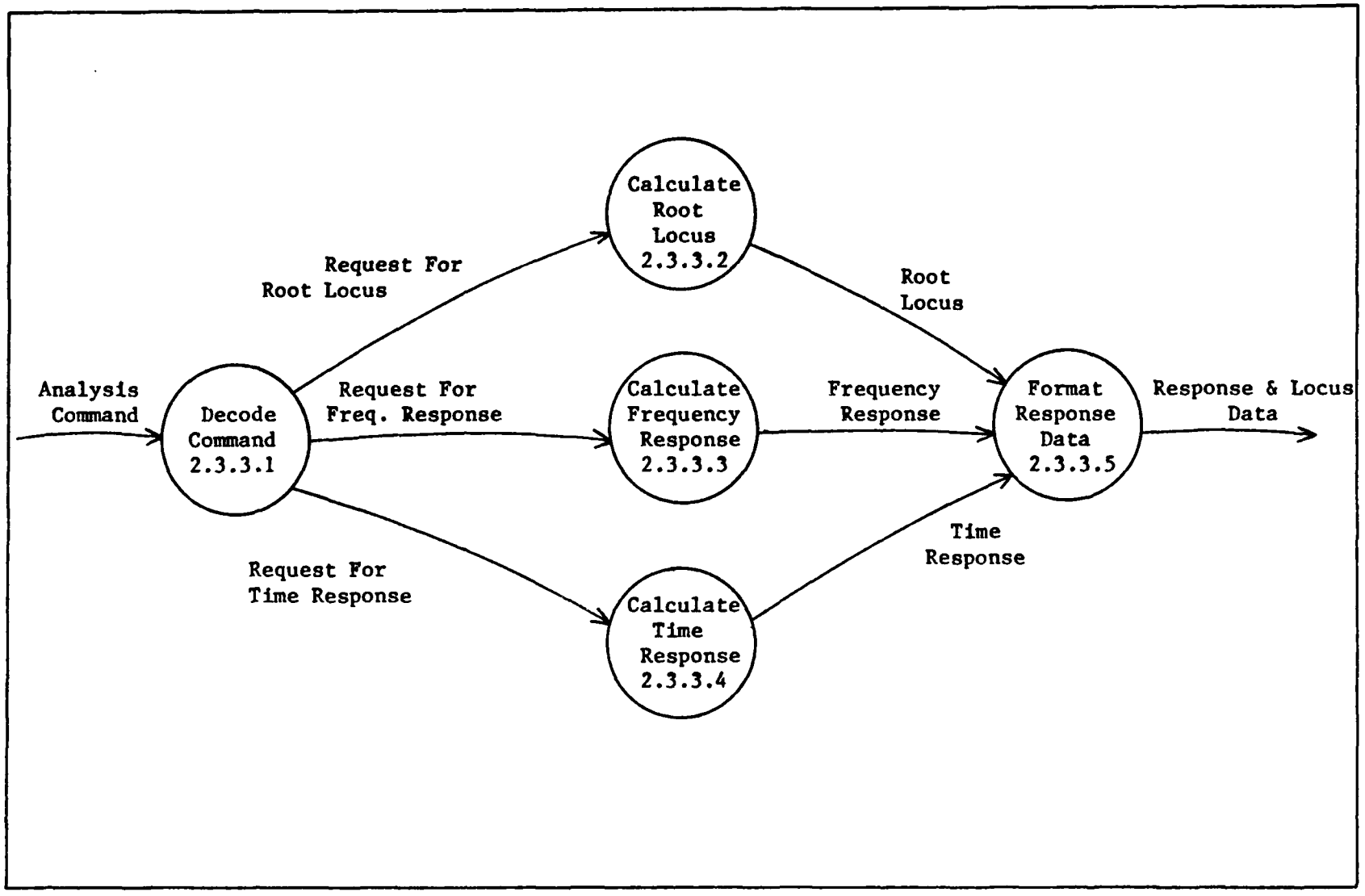


Figure A-10. Perform Conventional Analysis (node 2.3.3)

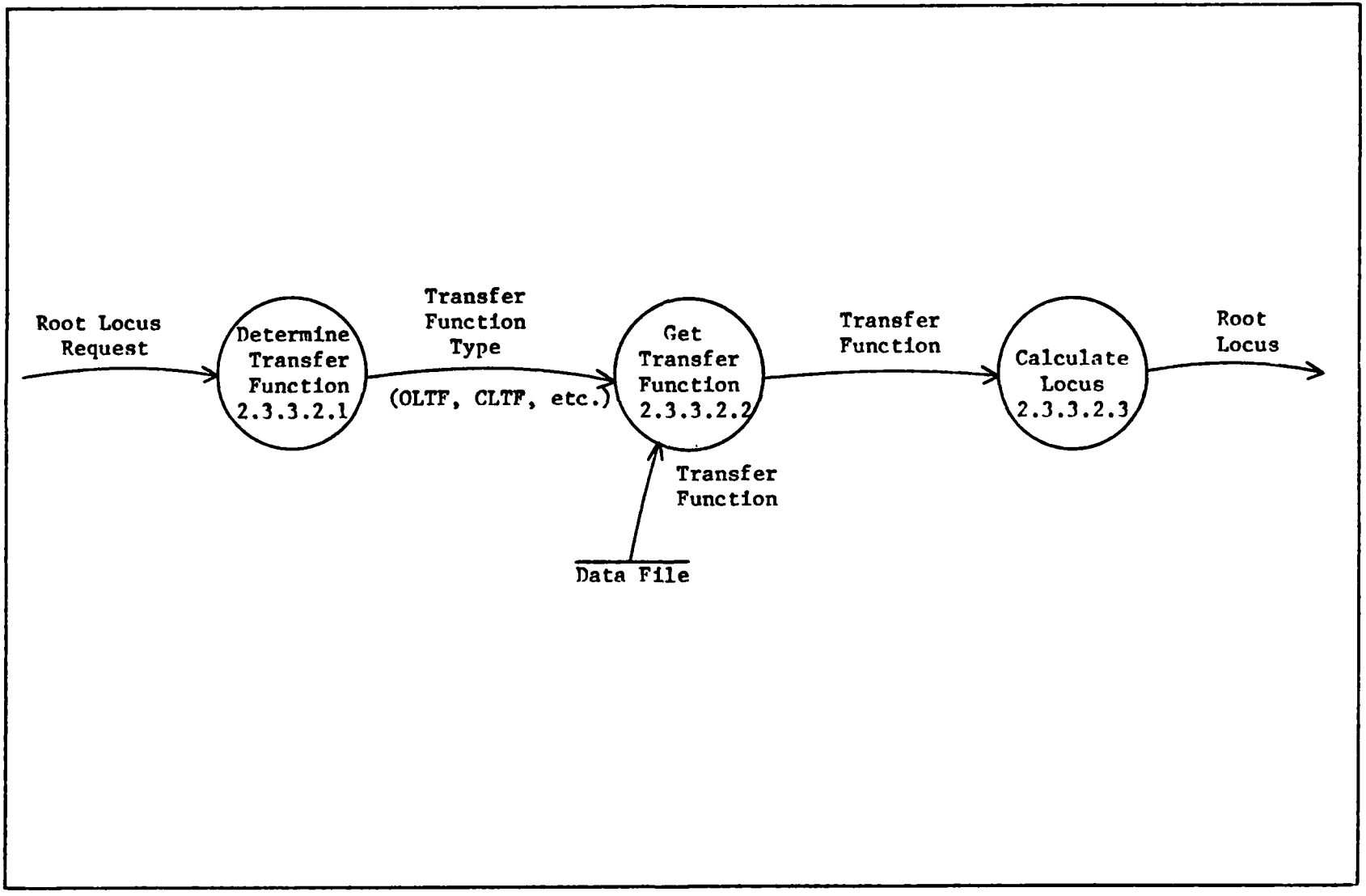


Figure A-11. Calculate Root Locus (node 2.3.3.2)

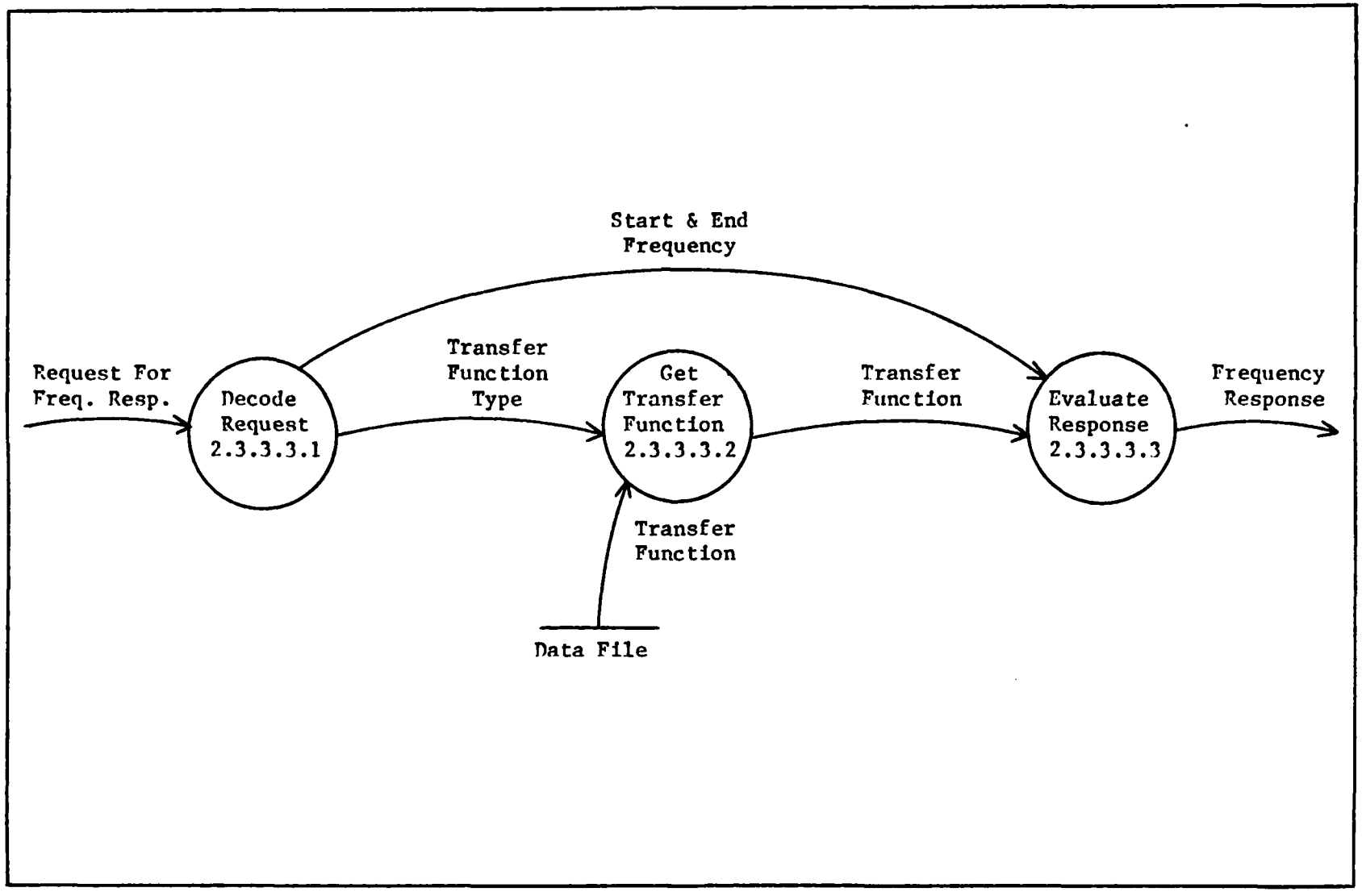


Figure A-12. Calculate Frequency Response (node 2.3.3.3)

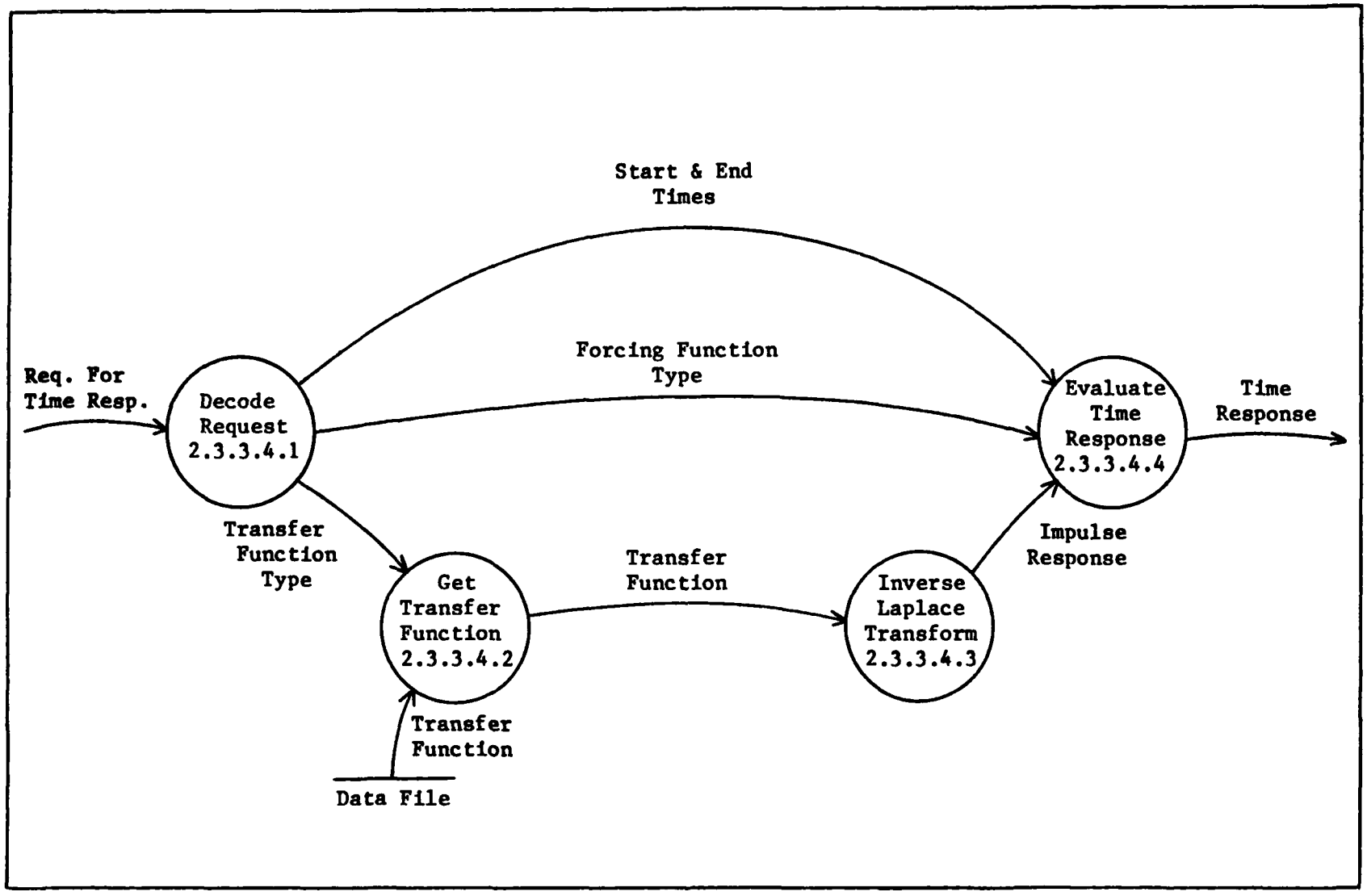


Figure A-13. Calculate Time Response (node 2.3.3.4)

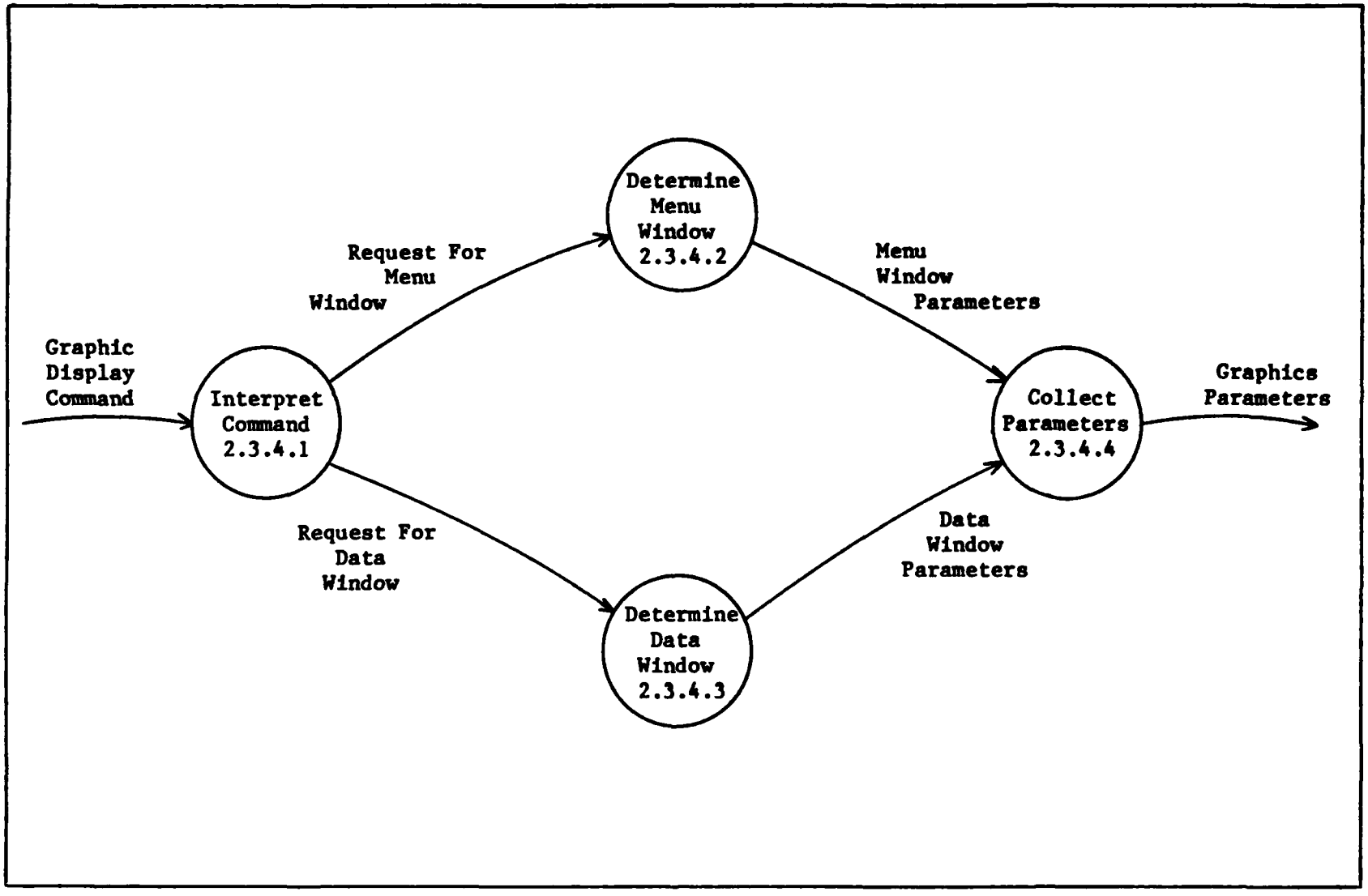


Figure A-14. Set Graphics Parameters (node 2.3.4)

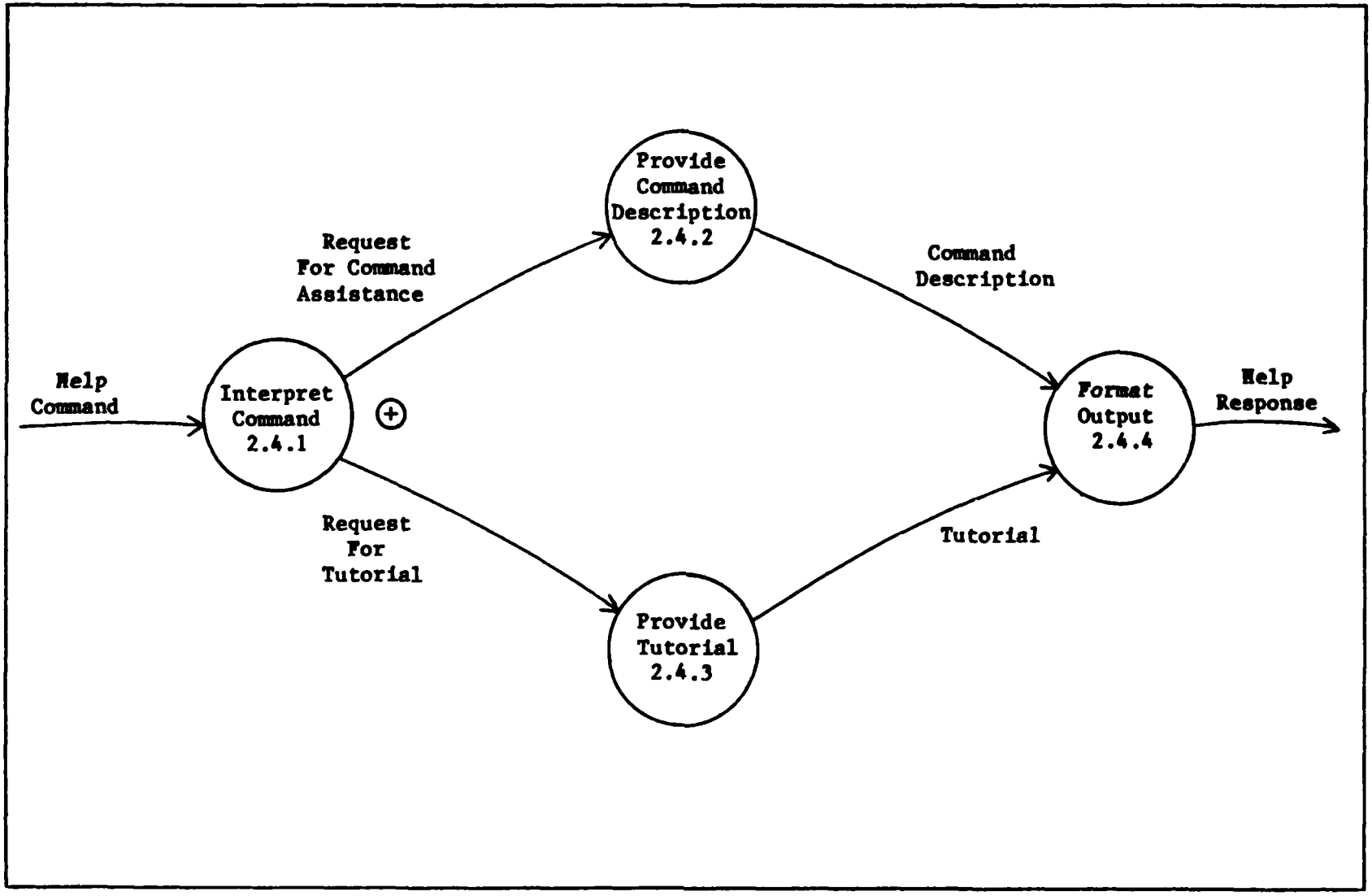


Figure A-15. Determine Help Needed (node 2.4)

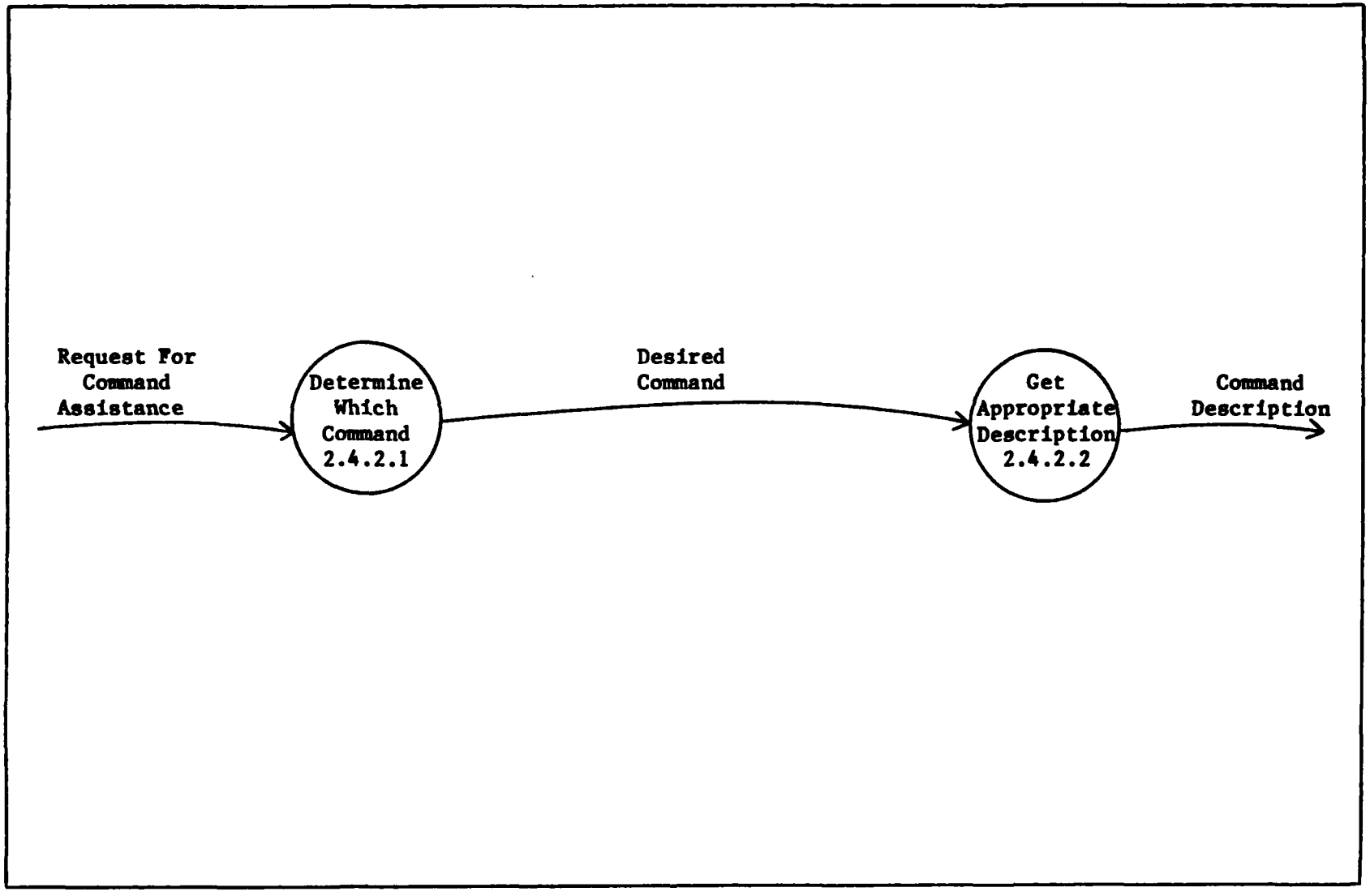


Figure A-16. Provide Command Description (node 2.4.2)

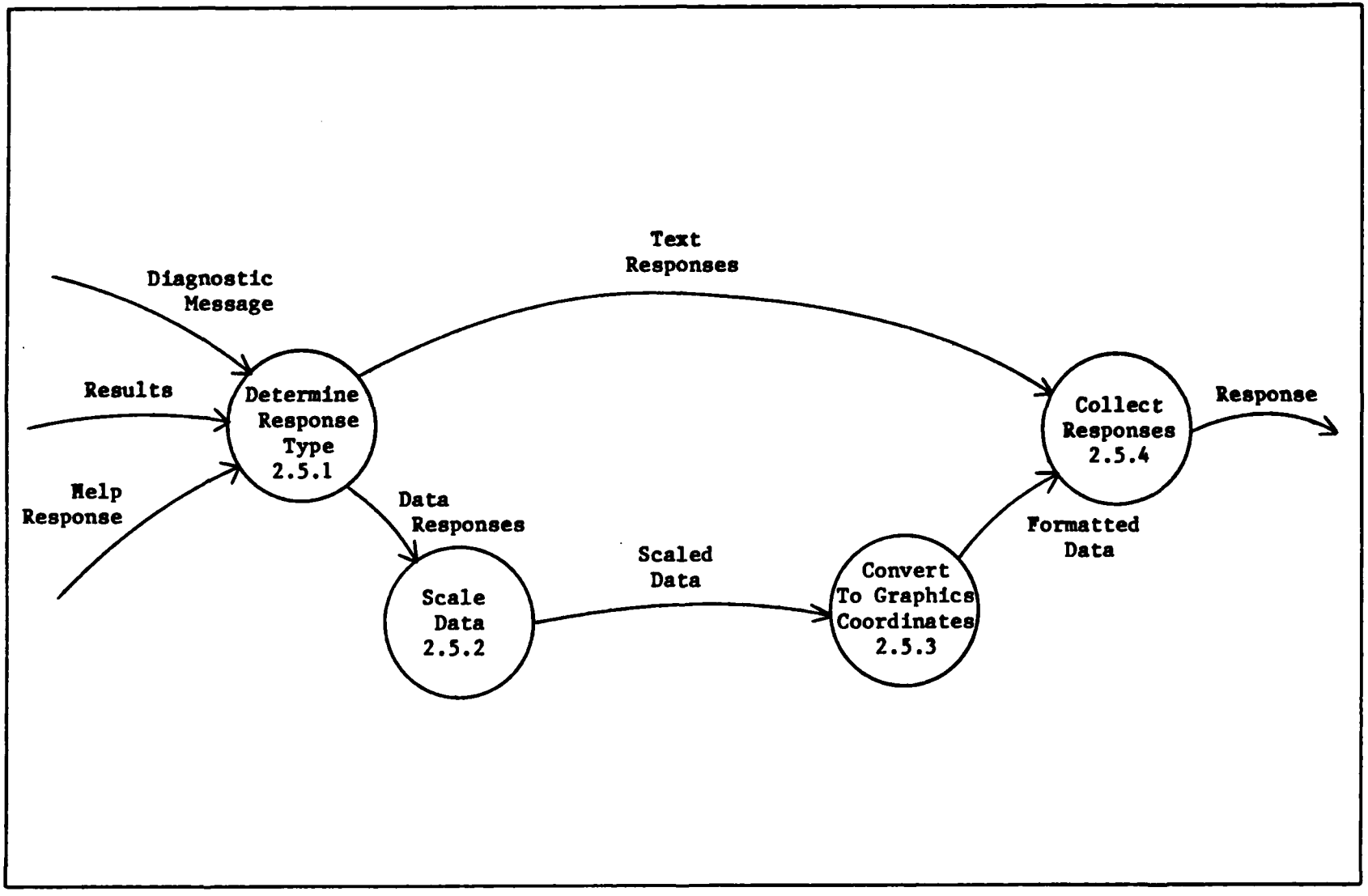


Figure A-17. Format Response (node 2.5)

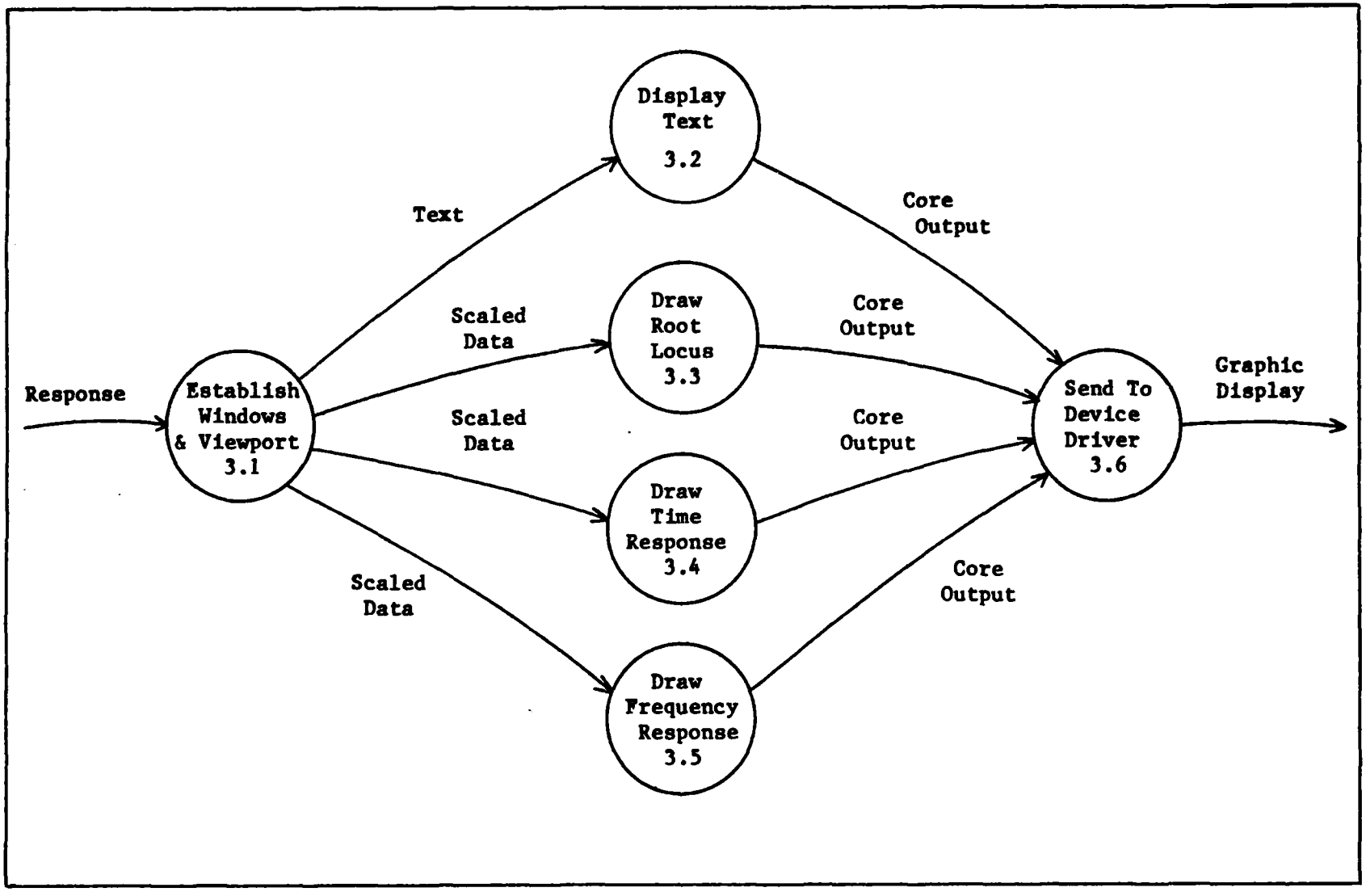


Figure A-18. Perform Graphic Functions (node 3)

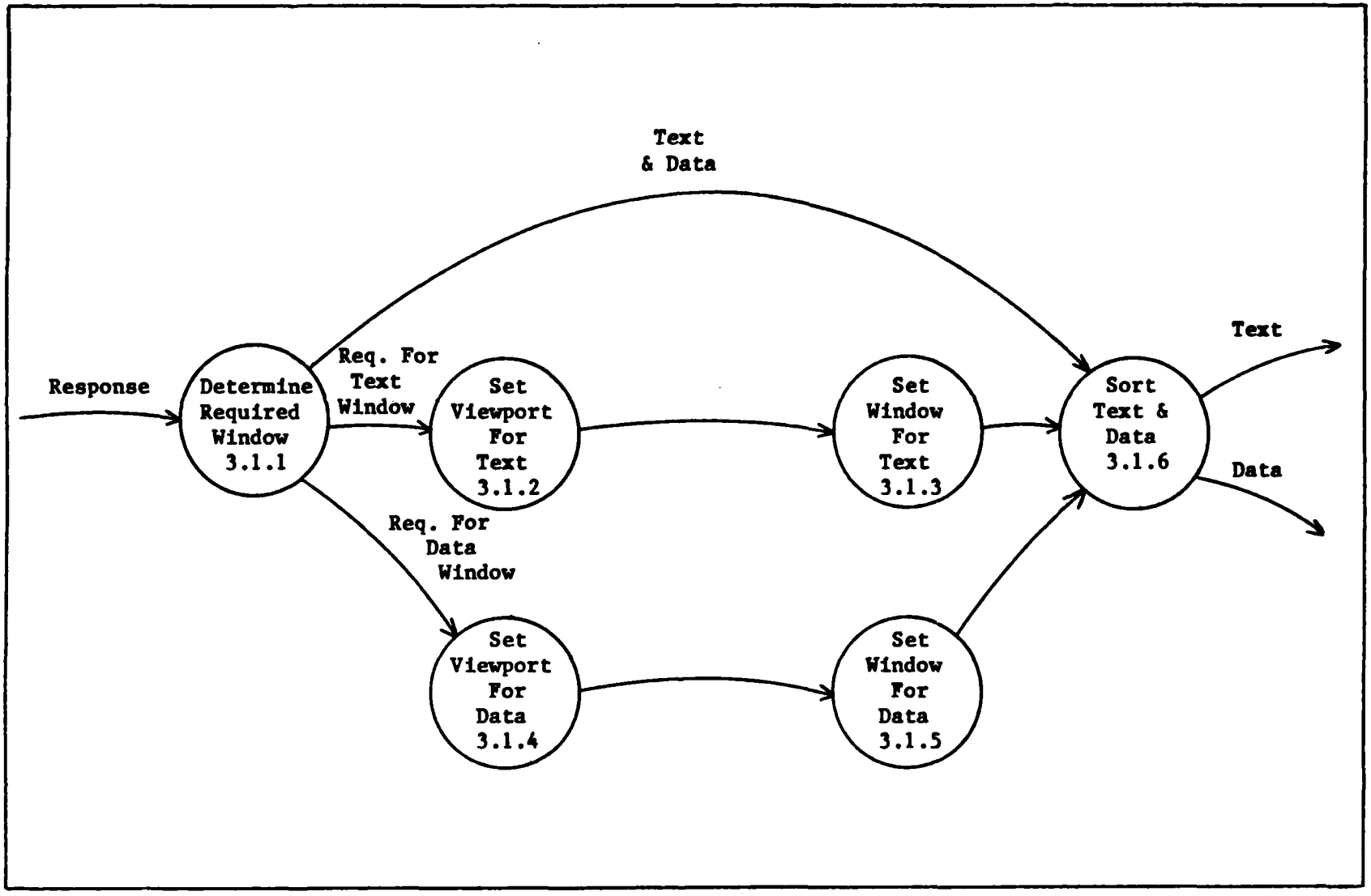


Figure A-19. Establish Windows & Viewports (node 3.1)

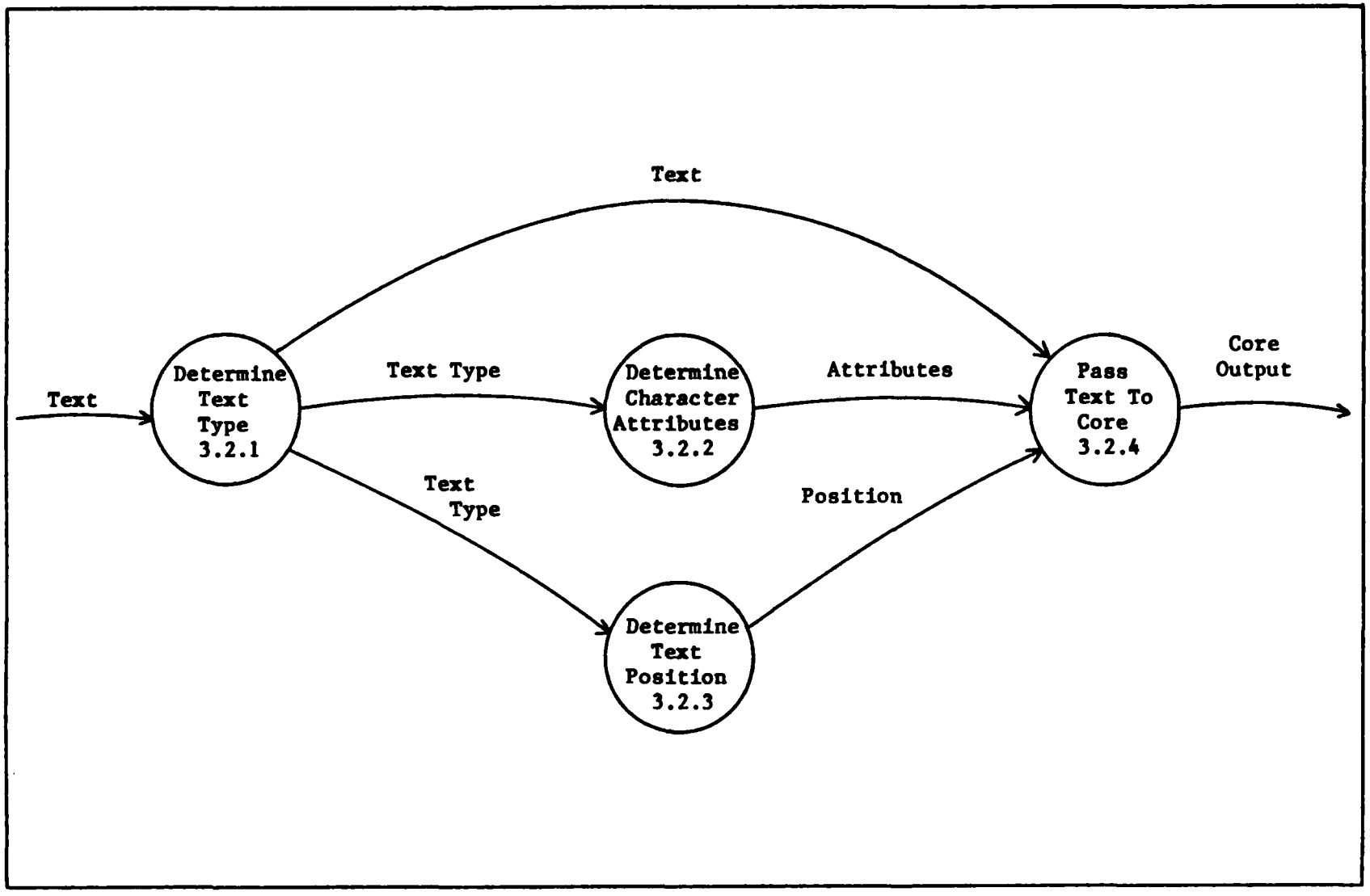


Figure A-20. Display Text (node 3.2)

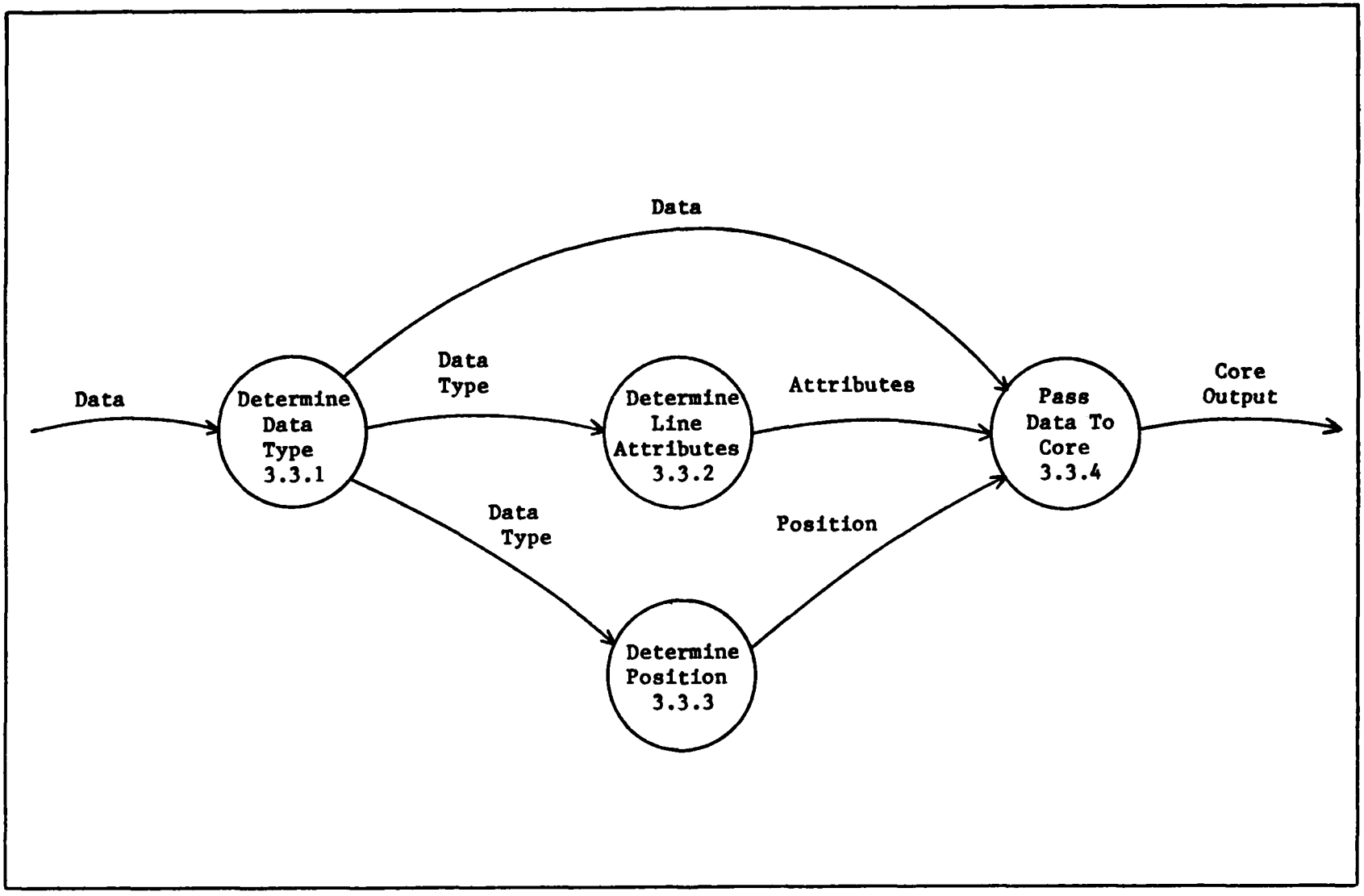


Figure A-21. Display Data (nodes 3.3 - 3.5)

STRUCTURE CHARTS

The structure charts which follow provide a more detailed view of the system hierarchy. From these diagrams, it is easy to determine which modules (subroutines or procedures) are called or used by a particular module to perform its function. In addition, it is easy to see on these diagrams all flows of data (open circles) and control (solid circles). To make these diagrams most useful, they have been updated to represent the final design. All additions or modifications to the initial design have been incorporated into these diagrams. All of the variable names on these diagrams are global names so that they will correspond to previous documentation for the GWCORE and VAX TOTAL. For example, all of the variables shown for GWCORE functions have the same name as found in the GWCORE user's document. This convention in variable names simplifies the relationship between this design document and other formal documentation.

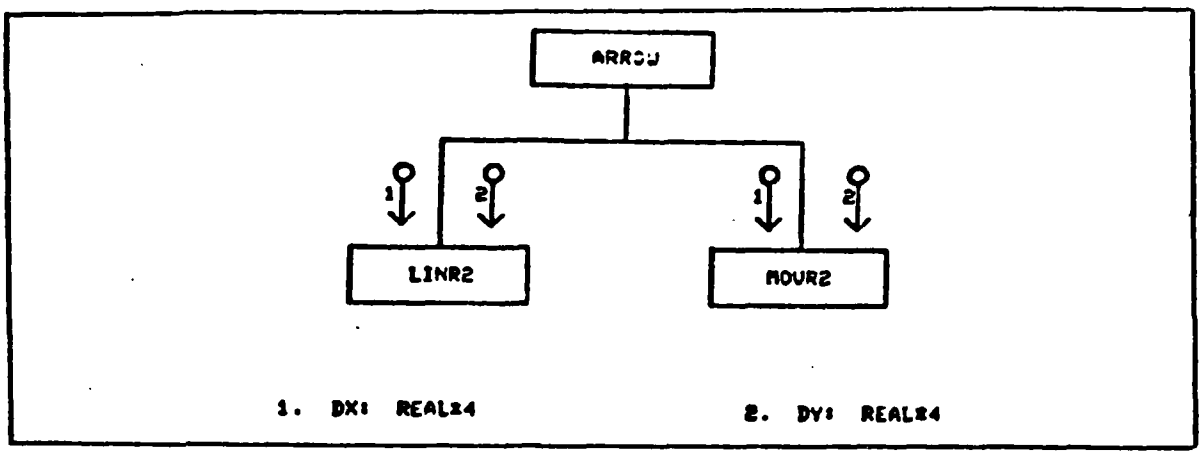


Figure A-22. Structure Chart for ARROW

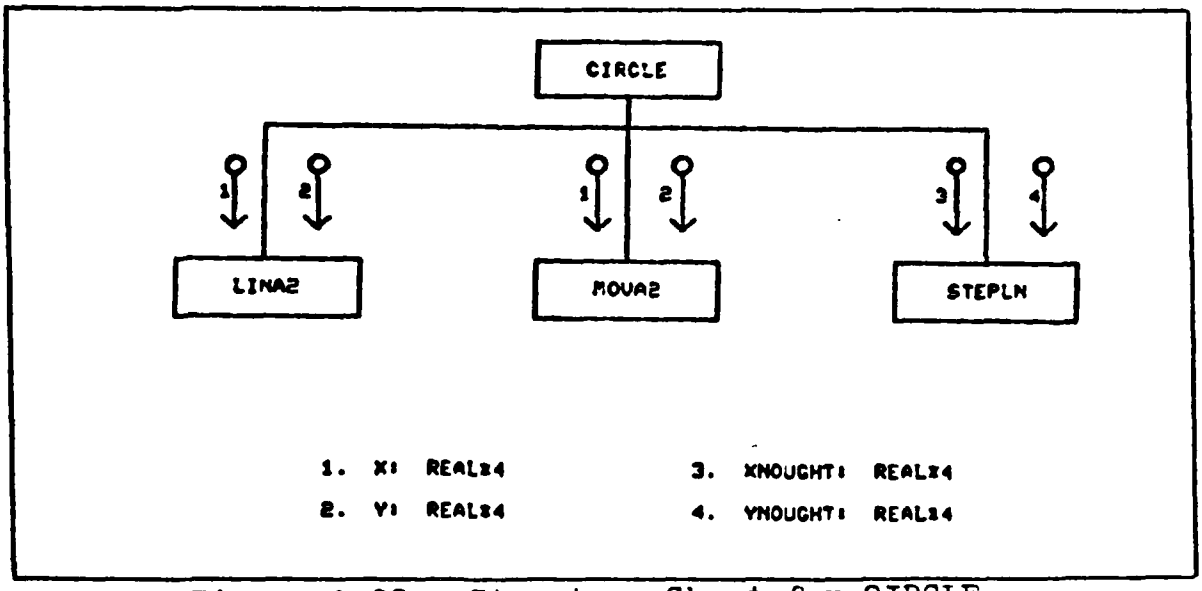


Figure A-23. Structure Chart for CIRCLE

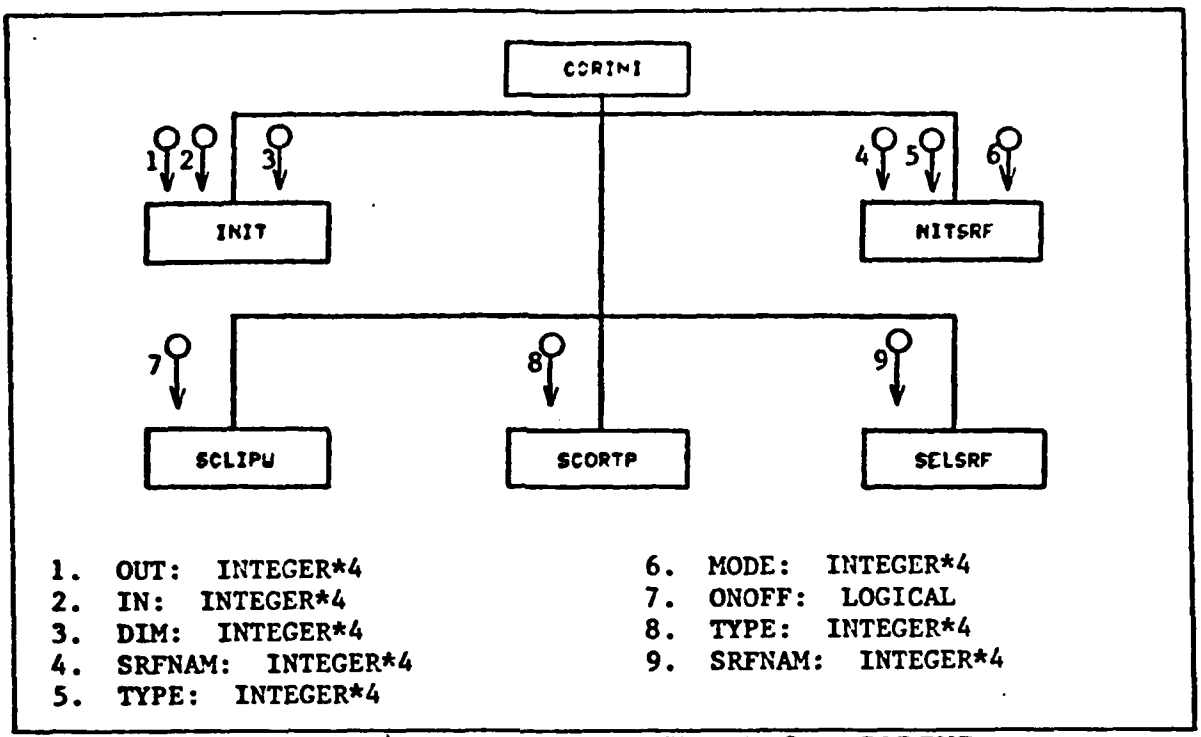


Figure A-24. Structure Chart for CORINI

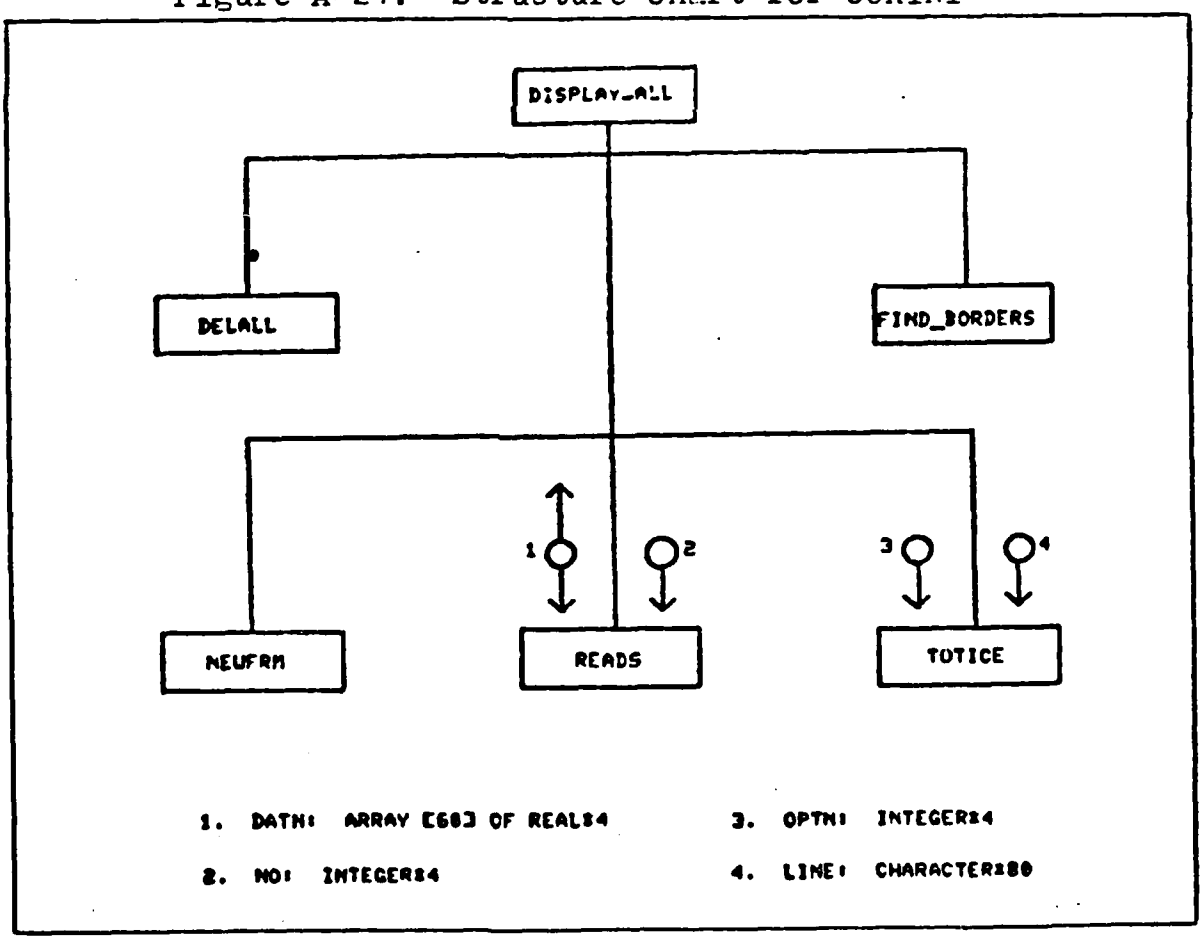


Figure A-25. Structure Chart for DISPLAY ALL

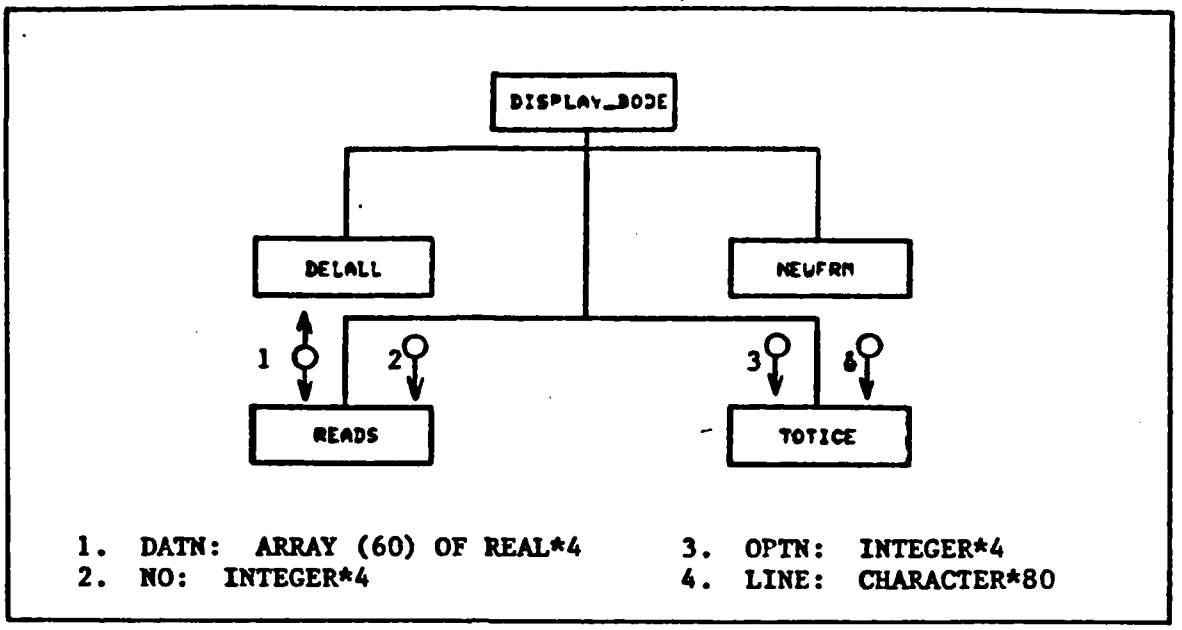
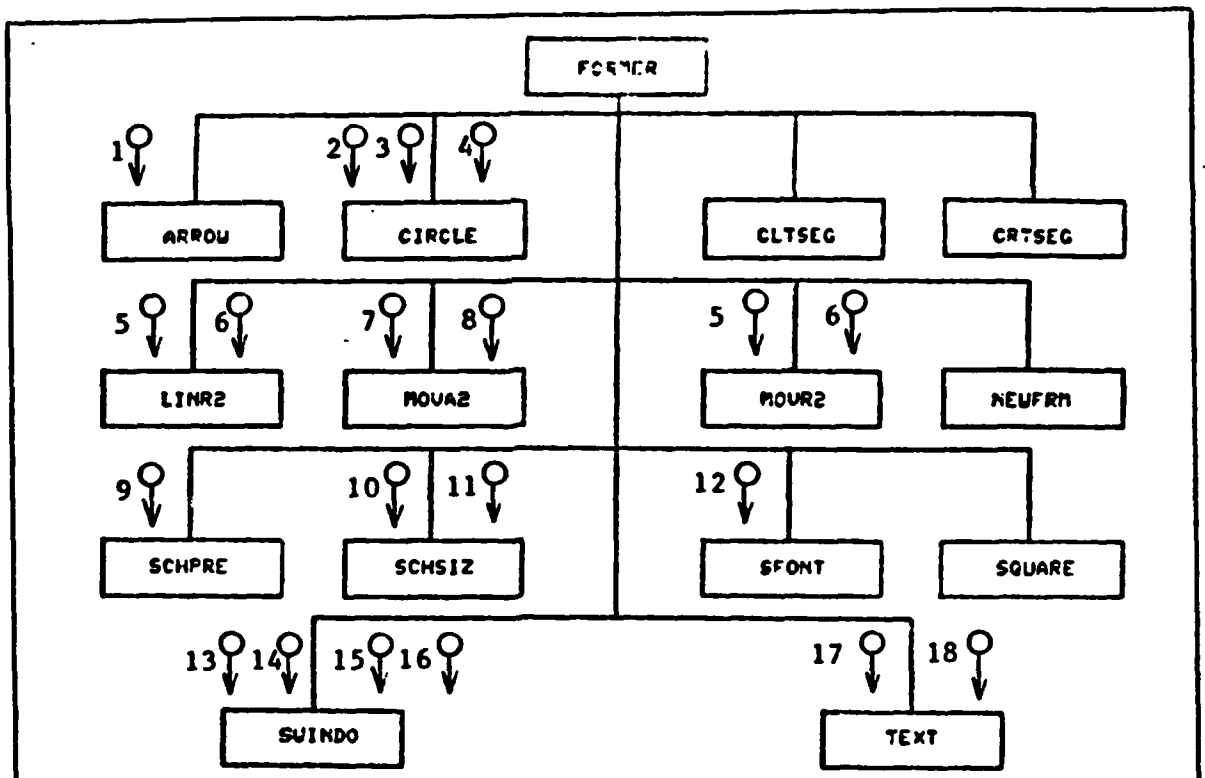


Figure A-26. Structure Chart for DISPLAY_BODE



1. KDIR: INTEGER*4
2. XNOUGHT: REAL*4
3. YNOUGHT: REAL*4
4. RADIUS: REAL*4
5. DX: REAL*4
6. DY: REAL*4
7. X: REAL*4
8. Y: REAL*4
9. CHPREC: INTEGER*4
10. CHWID: REAL*4
11. CHHY: REAL*4
12. FONT: INTEGER*4
13. XMIN: REAL*4
14. XMAX: REAL*4
15. YMIN: REAL*4
16. YMAX: REAL*4
17. CHARST: CHARACTER
18. N: INTEGER*4

Figure A-27. Structure Chart for FORMER

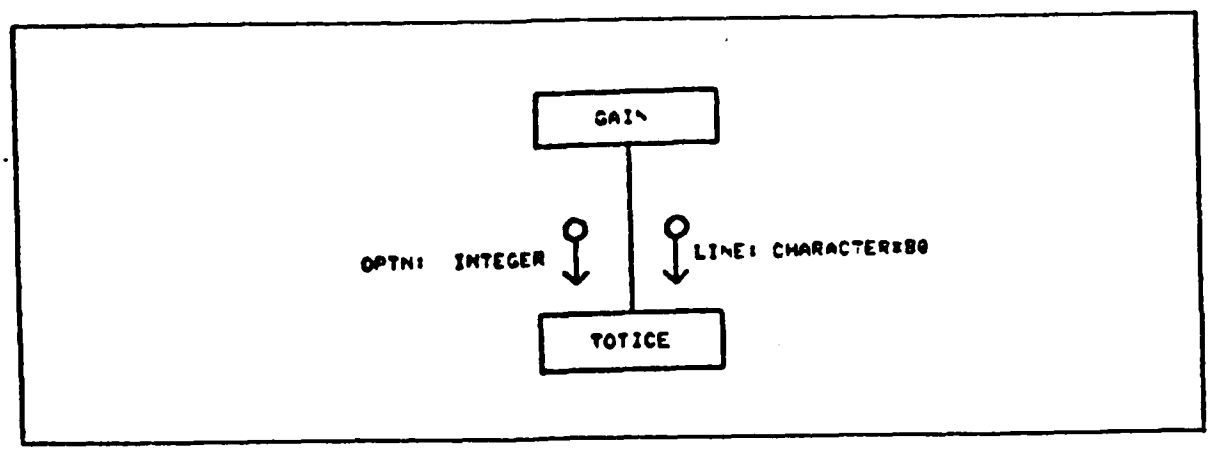
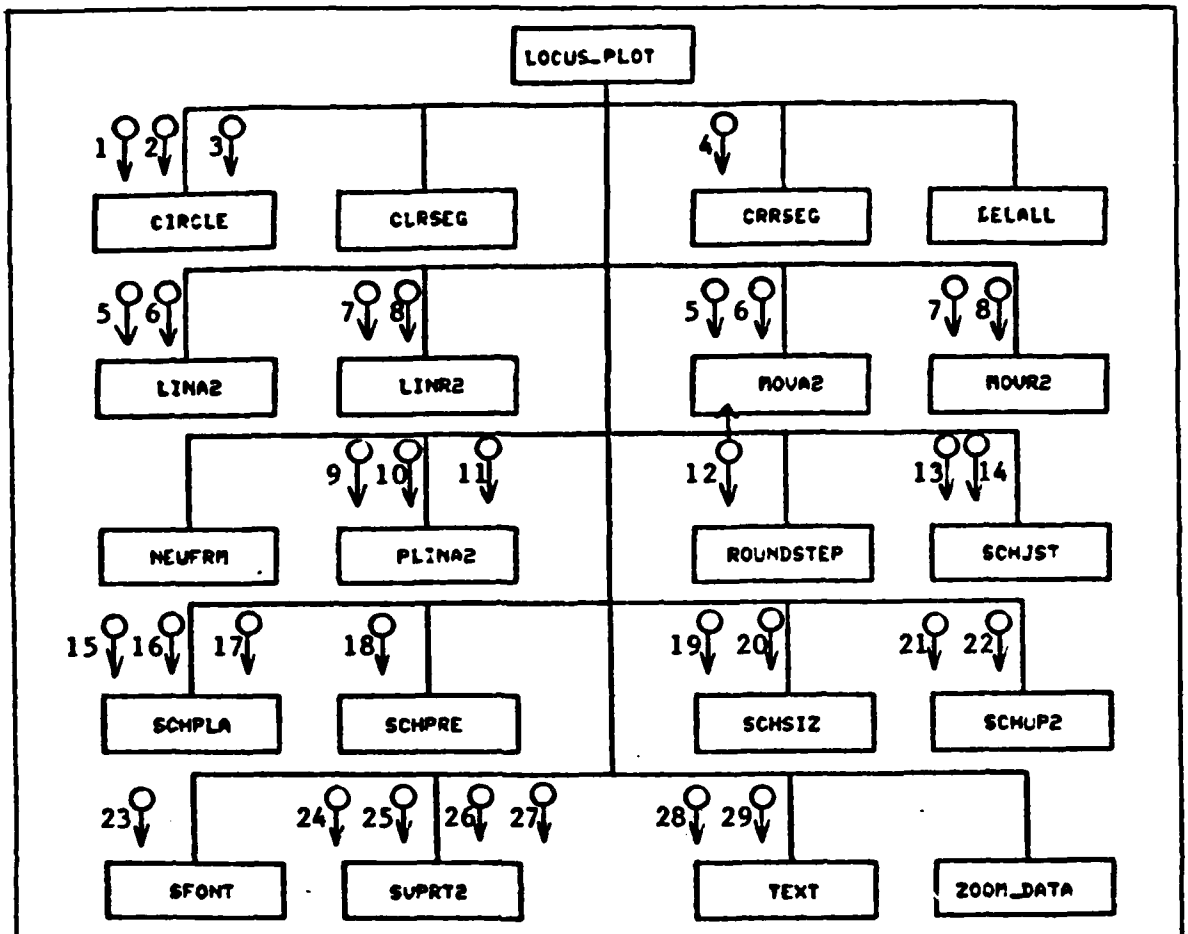


Figure A-28. Structure Chart for GAIN



1. XNOUGHT: REAL*4
2. YNOUGHT: REAL*4
3. RADIUS: REAL*4
4. SEGNAM: INTEGER*4
5. X: REAL*4
6. Y: REAL*4
7. DX: REAL*4
8. DY: REAL*4
9. XARAY: REAL*4
10. YARAY: REAL*4
11. N: INTEGER*4
12. STEPVAL: REAL*4
13. CHJSTH: INTEGER*4
14. CHJSTV: INTEGER*4
15. DX: REAL*4
16. DY: REAL*4
17. DZ: REAL*4
18. CHPREC: INTEGER*4
19. CHWID: REAL*4
20. CHHY: REAL*4
21. DXCHUP: REAL*4
22. DYCHUP: REAL*4
23. FONT: INTEGER*4
24. XM[illegible] REAL*4
25. XMAX: REAL*4
26. YMIN: REAL*4
27. YMAX: REAL*4
28. CHARST: CHARACTER
29. N: INTEGER

Figure A-29. Structure Chart for LOCUS_PLOT

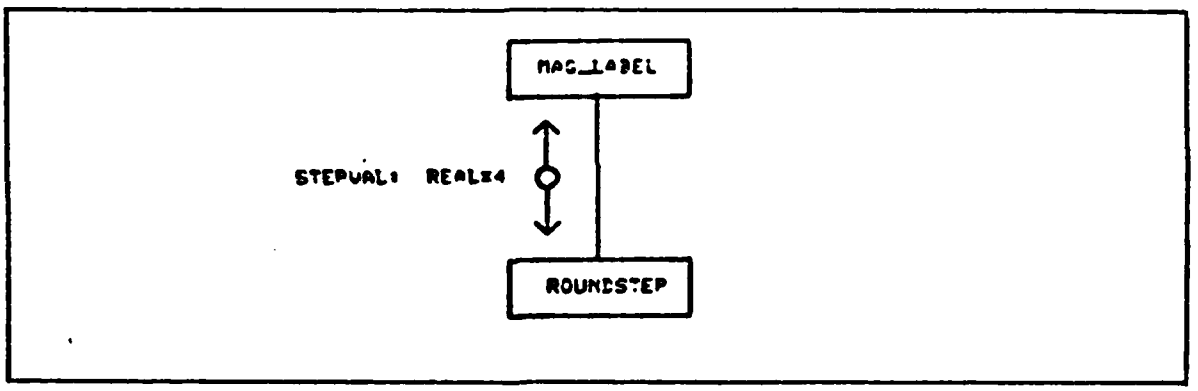
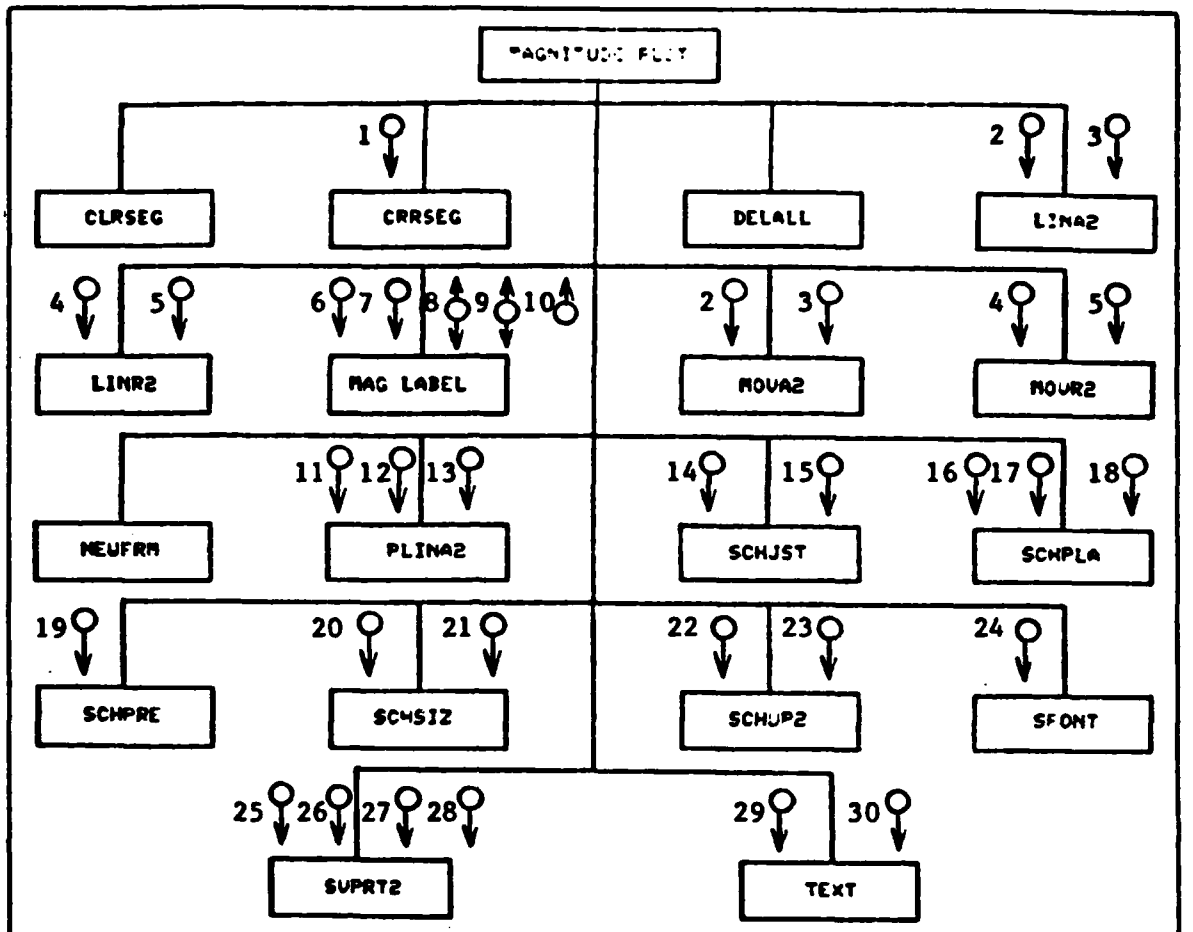


Figure A-30. Structure Chart for MAG_LABEL



1. SEGNAM: INTEGER*4
2. X: REAL*4
3. Y: REAL*4
4. DX: REAL*4
5. DY: REAL*4
6. XMIN: REAL*4
7. XMAX: REAL*4
8. YMIN: REAL*4
9. YMAX: REAL*4
10. YINC: REAL*4
11. XARAY: REAL*4
12. YARAY: REAL*4
13. N: INTEGER*4
14. CHJSTH: INTEGER*4
15. CHJSTV: INTEGER*4
16. DX: REAL*4
17. DY: REAL*4
18. DZ: REAL*4
19. CHPREC: INTEGER*4
20. CHWID: REAL*4
21. CHHY: REAL*4
22. DXCHUP: REAL*4
23. DYCHUP: REAL*4
24. FONT: INTEGER*4
25. XMIN: REAL*4
26. XMAX: REAL*4
27. YMIN: REAL*4
28. YMAX: REAL*4
29. CHARST: CHARACTER
30. N: INTEGER*4

Figure A-31. Structure Chart for MAGNITUDE_PLOT

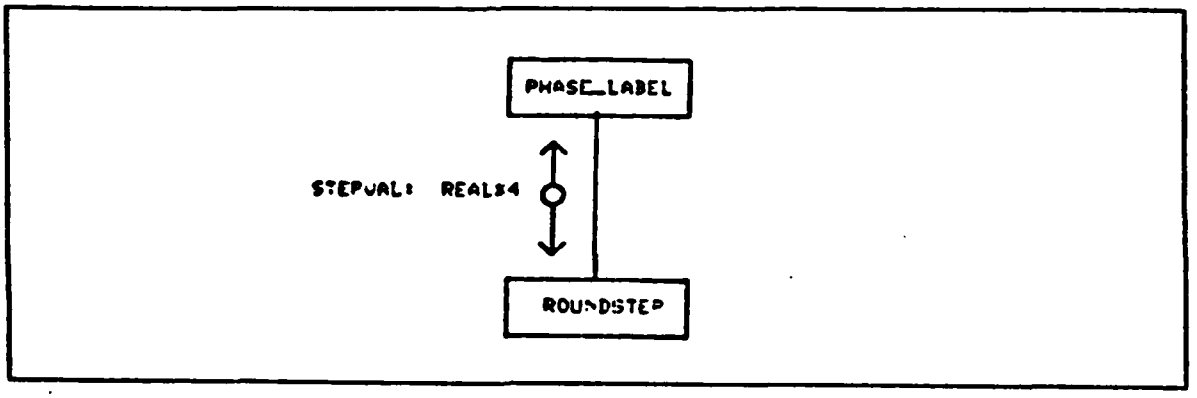
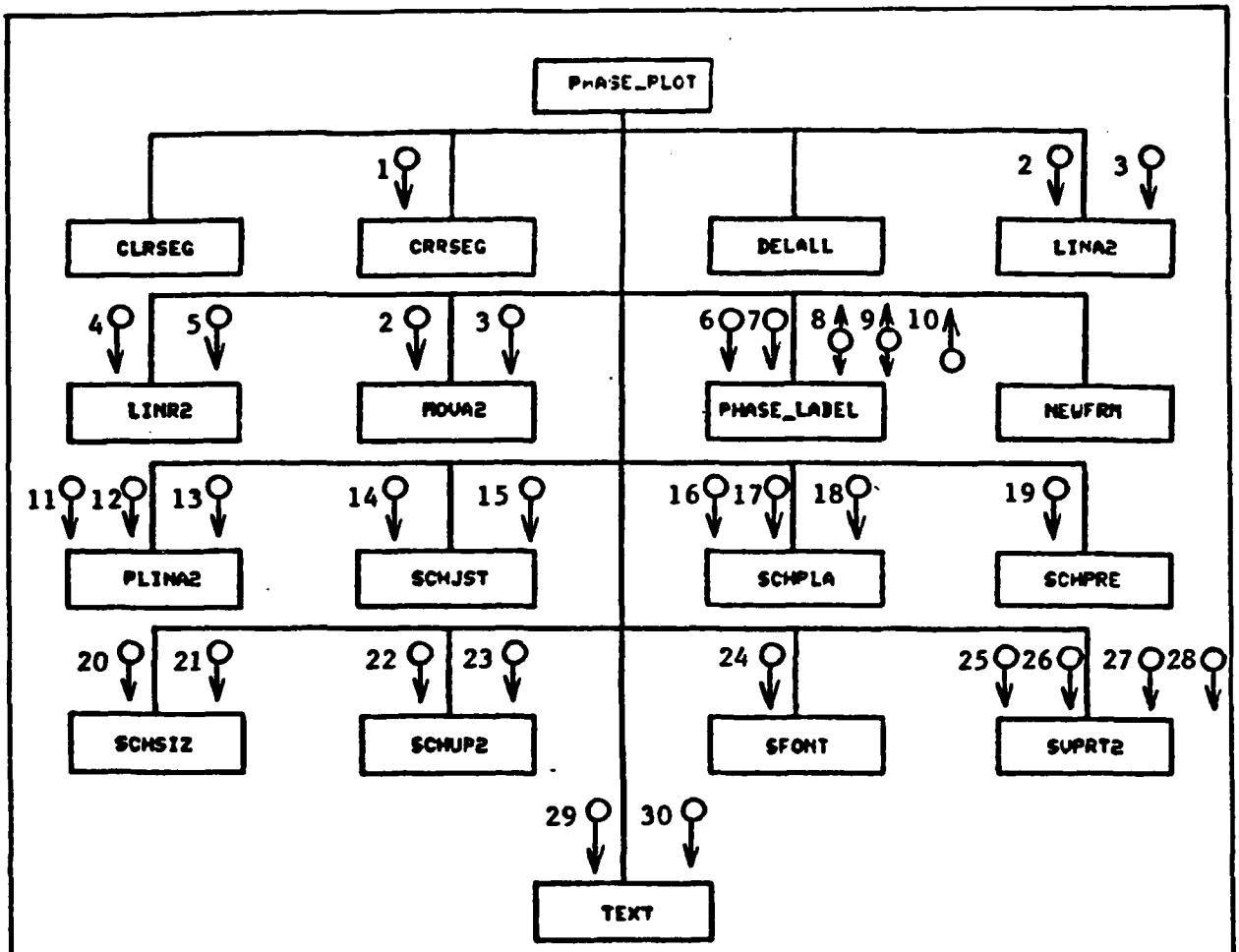


Figure A-32. Structure Chart for PHASE_LABEL



1. SEGNAM: INTEGER*4
2. X: REAL*4
3. Y: REAL*4
4. DX: REAL*4
5. DY: REAL*4
6. XMIN: REAL*4
7. XMAX: REAL*4
8. YMIN: REAL*4
9. YMAX: REAL*4
10. YINC: REAL*4
11. XARAY: REAL*4
12. YARAY: REAL*4
13. N: INTEGER*4
14. CHJSTH: INTEGER*4
15. CHJSTV: INTEGER*4
16. DX: REAL*4
17. DY: REAL*4
18. DZ: REAL*4
19. CHPREC: INTEGER*4
20. CHWID: REAL*4
21. CHHY: REAL*4
22. DXCHUP: REAL*4
23. DYCHUP: REAL*4
24. FONT: INTEGER*4
25. XMIN: REAL*4
26. XMAX: REAL*4
27. YMIN: REAL*4
28. YMAX: REAL*4
29. CHARST: CHARACTER
30. N: INTEGER*4

Figure A-33. Structure Chart for PHASE_PLOT

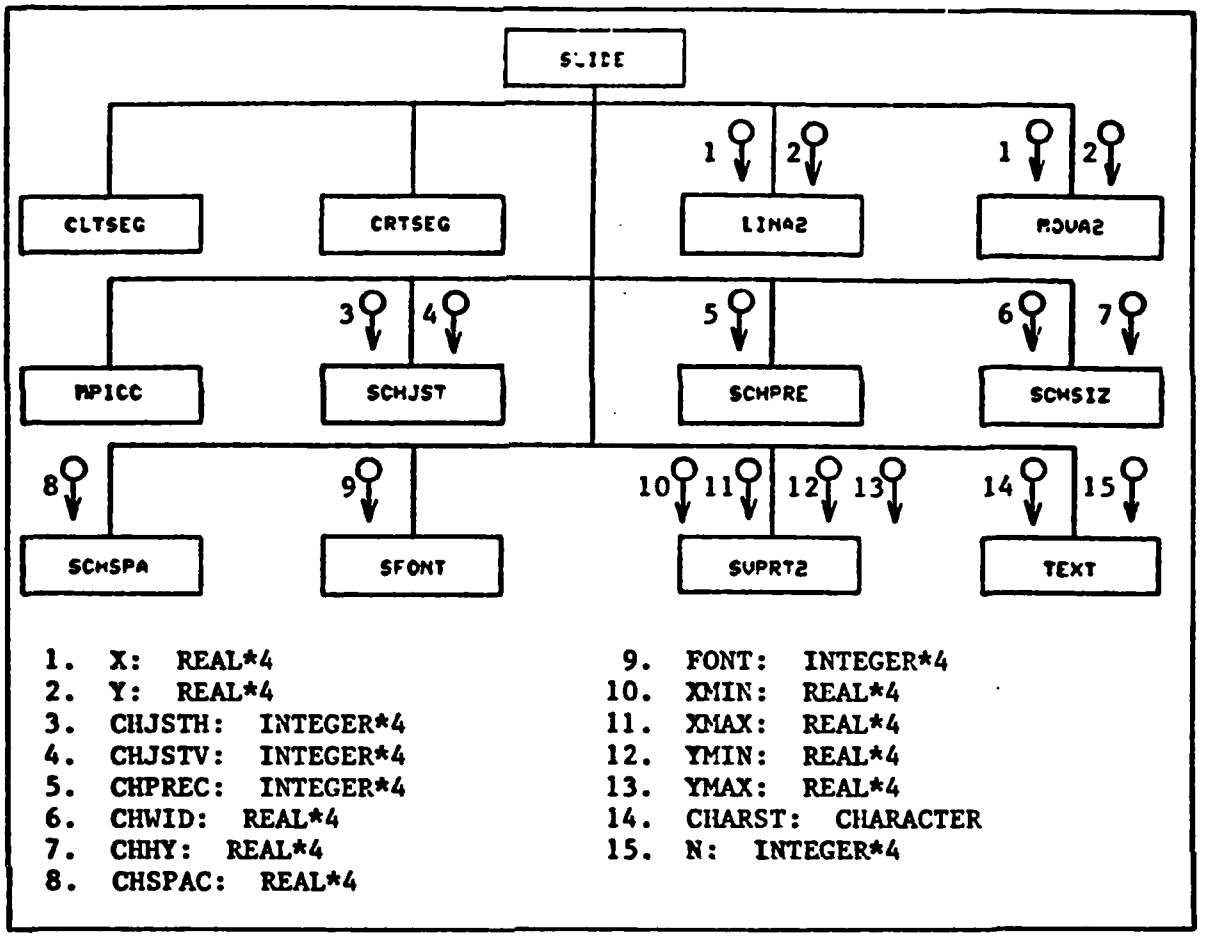


Figure A-34. Structure Chart for SLIDE

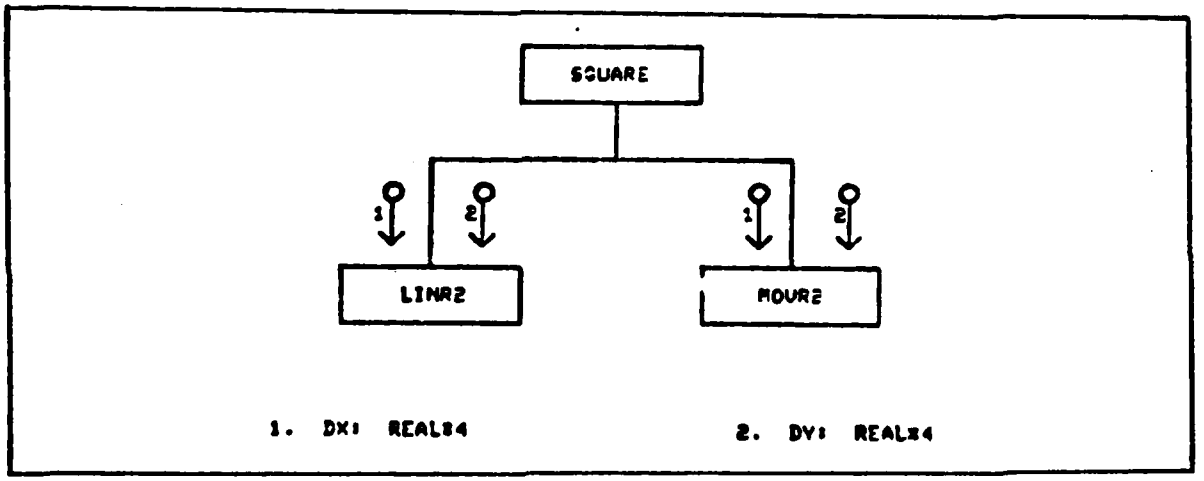


Figure A-35. Structure Chart for SQUARE

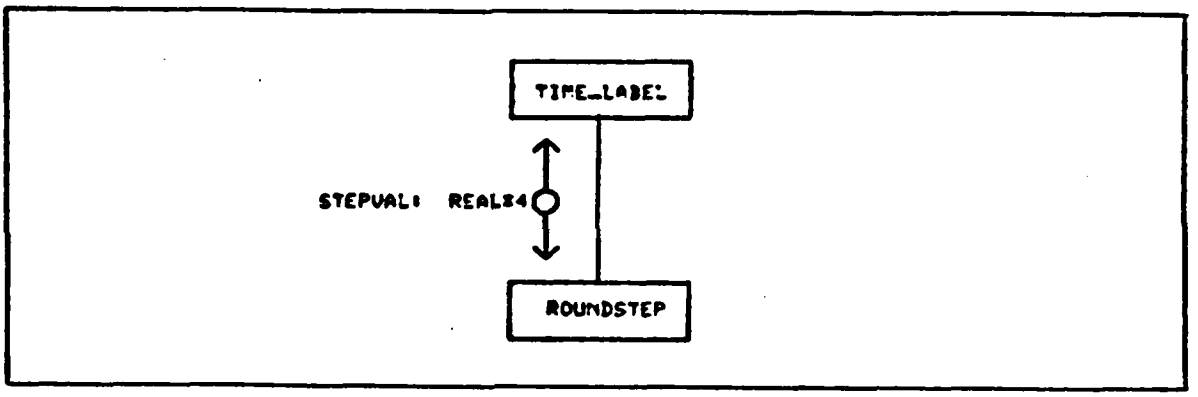
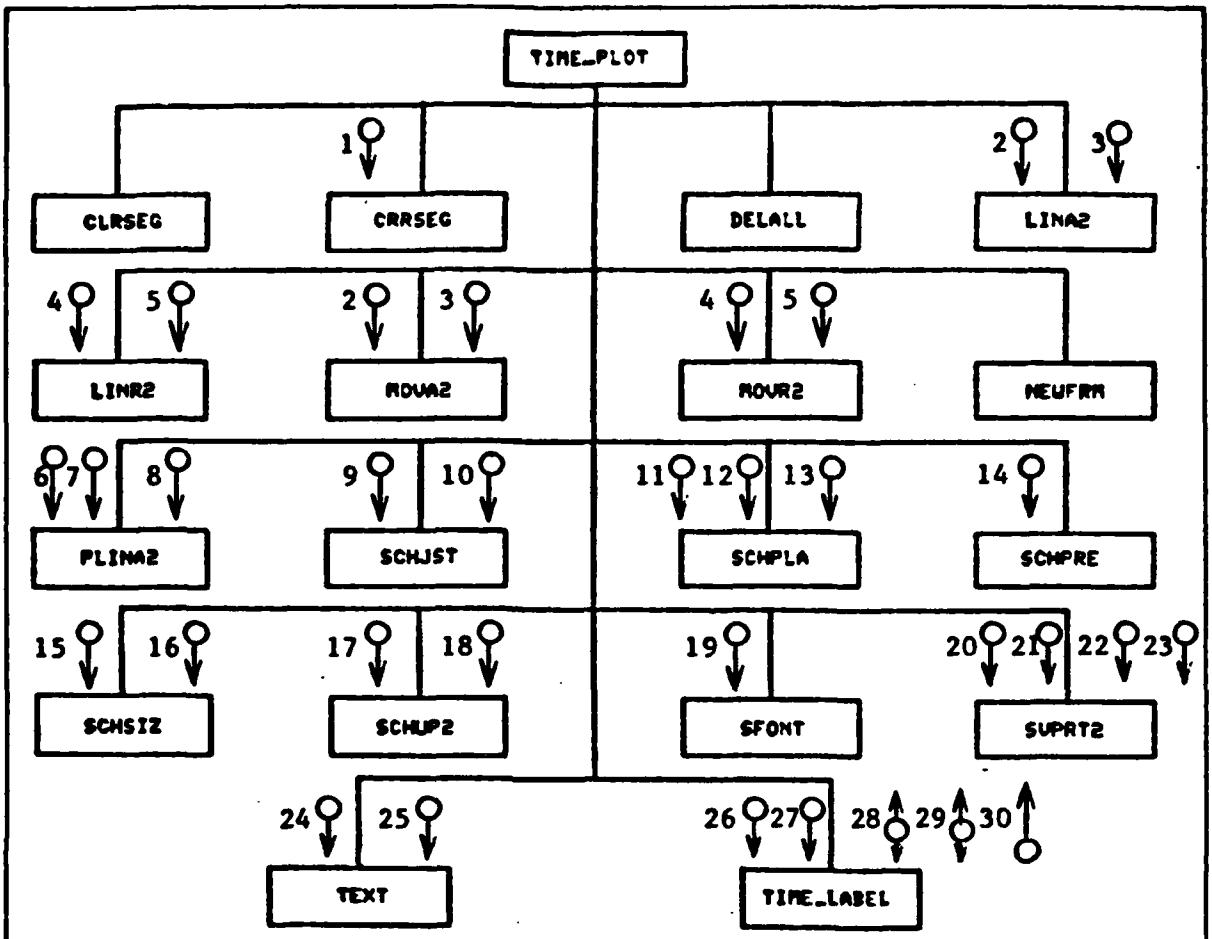


Figure A-36. Structure Chart for TIME_LABEL



1. SEGNAM: INTEGER*4
2. X: REAL*4
3. Y: REAL*4
4. DX: REAL*4
5. DY: REAL*4
6. XARAY: REAL*4
7. YARAY: REAL*4
8. N: INTEGER*4
9. CHJSTH: INTEGER*4
10. CHJSTV: INTEGER*4
11. DX: REAL*4
12. DY: REAL*4
13. DZ: REAL*4
14. CHPREC: REAL*4
15. CHWID: REAL*4
16. CHHY: REAL*4
17. DXCHUP: REAL*4
18. DYCHUP: REAL*4
19. FONT: INTEGER*4
20. XMIN: REAL*4
21. XMAX: REAL*4
22. YMIN: REAL*4
23. YMAX: REAL*4
24. CHARST: CHARACTER
25. N: INTEGER*4
26. XMIN: REAL*4
27. XMAX: REAL*4
28. YMIN: REAL*4
29. YMAX: REAL*4
30. YINC: REAL*4

Figure A-37. Structure Chart for TIME_PLOT

DATA DICTIONARIES

The data dictionaries for ICECAP-II follow this paragraph. Rather than complete data element descriptions, these are a type of condensed or abbreviated dictionary. A table for each new module lists all the variables used within the module, the type of the variable, and a brief description. This type of dictionary is used in ICECAP-II primarily because Gembarowski used a similar format to document the original ICECAP. Thus, a complete dictionary with consistent formats can be obtained by combining Gembarowski's original tables with those which follow.

SUBROUTINE: ARROW

FILE NAME: ARROW.FOR

LANGUAGE: VAX-11 FORTRAN

DESCRIPTION: THIS MODULE USES THE GWCORE TO DRAW ARROWHEADS FOR THE BLOCK DIAGRAMS.

CALLING MODULES: FORMER

CALLING PROCEDURE:
CALL ARROW (KDIR)

PASSED ARGUMENT: KDIR

MODULES CALLED: LINR2 MOVR2

VARIABLES:	TYPE:	DESCRIPTION:
DEBUG	LOGICAL	INDICATES DEBUG SELECT
KDIR	INTEGER*4	DIRECTION OF ARROW

Table A-1. Dictionary for ARROW

SUBROUTINE: CIRCLE

FILE NAME: CIRCLE.FOR

LANGUAGE: VAX-11 FORTRAN

DESCRIPTION: USES THE GWCORE TO DRAW A CIRCLE GIVEN THE COORDINATES OF THE CENTER AND RADIUS. (DEVELOPED BY KEVIN ROSE)

CALLING MODULES: FORMER, LOCUS_PLOT

CALLING PROCEDURE:
CALL CIRCLE (XNOUGHT, YNOUGHT, RADIUS)

PASSED ARGUMENTS: XNOUGHT, YNOUGHT, RADIUS

MODULES CALLED: LINA2 MOVA2 STEPLN

VARIABLES:	TYPE:	DESCRIPTION:
DEBUG	LOGICAL	INDICATES DEBUG SELECT
K	REAL*4	PLOT INCREMENT
RADIUS	REAL*4	RADIUS OF CIRCLE
XN	REAL*4	INTERMEDIATE X VALUE
XNOUGHT	REAL*4	X COORD. OF CENTER
XP	REAL*4	INTERMEDIATE X VALUE
YN	REAL*4	INTERMEDIATE Y VALUE
YNOUGHT	REAL*4	Y COORD. OF CENTER
YP	REAL*4	INTERMEDIATE Y VALUE

Table A-2. Dictionary for CIRCLE

SUBROUTINE: CORINI

FILE NAME: CORINI.FOR

LANGUAGE: VAX-11 FORTRAN

DESCRIPTION: INITIALIZES THE GWCORE AND THE TEKTRONIX 4014 DEVICE DRIVER.

CALLING MODULES: ICER

CALLING PROCEDURE: CALL CORINI

MODULES CALLED: INIT NITSRF SCLIPW
SCORTP SELSRF

Table A-3. Dictionary for CORINI

SUBROUTINE: DISPLAY_ALL

FILE NAME: ALL.FOR

LANGUAGE: VAX-11 FORTRAN

DESCRIPTION: BASIC CONTROL ROUTINE FOR THE DISPLAY OPTION WHICH PLOTS ALL FOUR RESPONSES SIMULTANEOUSLY.

CALLING MODULES: DISPLAY_OR_PRIN

CALLING PROCEDURE: CALL DISPLAY_ALL

MODULES CALLED: DELALL, FIND_BORDERS, NEWFRM, READS, TOTICE

VARIABLES:	TYPE:	DESCRIPTION:
ALLFLAG	LOGICAL	INDICATES DISPLAY_ALL
BLANK	CHAR*80	EMPTY STRING
CLOSED	LOGICAL	CLOSED LOOP MODE
DATIN(2)	REAL*4	INPUT DATA
DEBUG	LOGICAL	INDICATES DEBUG SELECTED
DUMMY	REAL*4	USED FOR PAUSE
FILE(30,000)	INTEGER*4	PDF
FLEND	INTEGER*4	LENGTH OF PDF
FLPNTR	INTEGER*4	PDF POINTER
IPOW	INTEGER*4	POWER OF START FREQUENCY
NCYC	INTEGER*4	NO OF PLOT CYCLES
NGO	INTEGER*4	FORTRAN OUTPUT UNIT
TFF	REAL*4	END TIME OF PLOT
TTT	REAL*4	START TIME OF PLOT
YIN(2)	REAL*4	INPUT DATA

Table A-4. Dictionary for DISPLAY_ALL

SUBROUTINE: DISPLAY_BODE

FILE NAME: BODE.FOR

LANGUAGE: VAX-11 FORTRAN

DESCRIPTION: PLOTS THE MAGNITUDE AND PHASE SIMULTANEOUSLY.

CALLING MODULES: DISPLAY_OR_PRIN

CALLING PROCEDURE: CALL DISPLAY_BODE

MODULES CALLED: DELALL NEWFRM READS TOTICE

VARIABLES:	TYPE:	DESCRIPTION:
BLANK	CHAR*80	EMPTY STRING
BODFLAG	LOGICAL	INDICATES DISPLAY BODE
CLOSED	LOGICAL	CLOSED LOOP MODE
DEBUG	LOGICAL	INDICATES DEBUG SELECT
DUMMY	REAL*4	DUMMY INPUT VARIABLE
FILE(30000)	INTEGER*4	PSEUDO DISPLAY FILE
FLEND	INTEGER*4	LENGTH OF PDF
FLPNTR	INTEGER*4	INDEX FOR PDF
IPOW	INTEGER*4	POWER OF START FREQ.
KK	INTEGER*4	COUNTER
NCYC	INTEGER*4	NO. OF LOG CYCLES
NGO	INTEGER*4	FORTRAN OUTPUT UNIT
YIN(2)	REAL*4	INPUT DATA

Table A-5. Dictionary for DISPLAY_BODE

SUBROUTINE: FORMER

FILE NAME: FORMER.FOR

LANGUAGE: VAX-11 FORTRAN

DESCRIPTION: DRAWS ICECAP-II'S VALID BLOCK DIAGRAMS IN RESPONSE TO AN INVALID FORM COMMAND.

CALLING MODULES: FORM

CALLING PROCEDURE: CALL FORMER

MODULES CALLED:

ARROW	CIRCLE	CLTSEG	CRTSEG
LINR2	MOVA2	MOVR2	NEWFRM
SCHPRE	SCHSIZ	SFONT	SQUARE
SWINDO	TEXT		

VARIABLES:	TYPE:	DESCRIPTION:
DEBUG	LOGICAL	INDICATES DEBUG SELECT

Table A-6. Dictionary for FORMER

SUBROUTINE: GAIN

FILE NAME: GAIN.FOR

LANGUAGE: VAX-11 FORTRAN

DESCRIPTION: PROMPTS USER FOR A VALUE OF GAIN AND SENDS VALUE TO TOTICE.

CALLING MODULES: DEFINE

CALLING PROCEDURE: CALL GAIN

MODULES CALLED: TOTICE

VARIABLES:	TYPE:	DESCRIPTION:
DEBUG	LOGICAL	INDICATES DEBUG SELECTED
INGAIN	CHAR*50	DESIRED GAIN
LABEL	CHAR*5	VARIABLE NAME STRING
FULL	CHAR*80	CONCATENATE LABEL/INGAIN

Table A-7. Dictionary for GAIN

SUBROUTINE: LOCUS_PLOT

FILE NAME: LOCPLT.FOR

LANGUAGE: VAX-11 FORTRAN

DESCRIPTION: PLOTS THE ROOT LOCUS WHICH IS CALCULATED BY ROOT 12.

CALLING MODULES: TOTICE

CALLING PROCEDURE: CALL LOCUS_PLOT

MODULES CALLED:

CIRCLE	CLRSEG	CRRSEG	DELALL
LINA2	LINR2	MOVA2	MOVR2
NEWFRM	PLINA2	ROUNDSTEP	SCHJST
SCHPLA	SCHPRE	SCHSIZ	SCHUP2
SFONT	SVPRT2	TEXT	ZOOM_DATA

VARIABLES:	TYPE:	DESCRIPTION:
AA	REAL*4	LOCUS RIGHT BORDER
ALLFLAG	LOGICAL	INDICATES DISPLAY_ALL
BB	REAL*4	LOCUS TOP BORDER
CC	REAL*4	LOCUS LEFT BORDER
DD	REAL*4	LOCUS BOTTOM BORDER
DEBUG	LOGICAL	INDICATES DEBUG SELECT
DUMMY	REAL*4	DUMMY INPUT VARIABLE
FILE(30000)	INTEGER*4	PSEUDO DISPLAY FILE
FLEND	INTEGER*4	LENGTH OF PDF
GRID	LOGICAL	INCLUDE GRID ON PLOTS
I	INTEGER*4	COUNTER
ICOUNT	INTEGER*4	LENGTH OF LOUCUS TRACE
INSTRG	CHAR*16	INPUT STRING
J	INTEGER*4	COUNTER
KK	INTEGER*4	COUNTER
LOCDAT(1000,2)	REAL*4	X, Y TRACE POINTS
M	INTEGER*4	NUMBER OF ZEROS
N	INTEGER*4	NUMBER OF POLES
NCHARS	INTEGER*4	LENGTH OF INSTRG
NGO	INTEGER*4	FORTRAN OUTPUT UNIT
PLMAX	REAL*4	TEMPORARY MAXIMUM VALUE
SECOND	LOGICAL	SECOND PASS OF ZOOM
SEGNAME	INTEGER*2	SEGMENT NAME
XCO	REAL*4	X COORD. FOR POLE OR ZERO
XLOCDAT(1000)	REAL*4	ARRAY OF X COORDS.
XMAX	REAL*4	MAXIMUM X VALUE
XMIN	REAL*4	MINIMUM X VALUE
XMRK	REAL*4	X COORD. FOR GRID MARK
XP(50)	REAL*4	X COORDS. OF POLES
XPOS	REAL*4	X COORD. FOR TIC MARK

Table A-8. Dictionary for LOCUS_PLOT

SUBROUTINE: LOCUS_PLOT (continued)

VARIABLES:	TYPE:	DESCRIPTION:
XSCALE	REAL*4	X AXIS SCALE
XSTEP	REAL*4	SCALED VALUE OF XSCALE
XZ(50)	REAL*4	X COORDS. OF ZEROS
XZERO	REAL*4	SCALED 0 VALUE (X AXIS)
YCO	REAL*4	Y COORD. FOR POLE OR ZERO
YLOCDAT(1000)	REAL*4	ARRAY OF Y COORDS.
YMAX	REAL*4	MAXIMUM Y VALUE
YMIN	REAL*4	MINIMUM Y VALUE
YMRK	REAL*4	Y COORD. FOR GRID MARK
YP(50)	REAL*4	Y COORDS. OF POLES
YPOS	REAL*4	Y COORD. FOR TIC MARK
YSCALE	REAL*4	Y AXIS SCALE
YSTEP	REAL*4	SCALED VALUE OF YSCALE
YZ(50)	REAL*4	Y COORDS. OF ZEROS
YZERO	REAL*4	SCALED 0 VALUE (Y AXIS)
ZOOM	LOGICAL	INDICATES ZOOM PROCESS

Table A-8. Dictionary for LOCUS_PLOT (cont'd)

SUBROUTINE: MAG_LABEL

FILE NAME: FRPLT.FOR

LANGUAGE: VAX-11 FORTRAN

DESCRIPTION: THIS MODULE DETERMINES THE LABEL AND GRID INTERVAL FOR THE FREQUENCY RESPONSE MAGNITUDE PLOT.

CALLING MODULES: MAGNITUDE_PLOT

CALLING PROCEDURE:
CALL MAG_LABEL (XMIN, XMAX, YMIN, YMAX, YINC)

PASSED ARGUMENTS: XMIN, XMAX, YMIN, YMAX, YINC
RETURNED ARGUMENTS: YMIN, YMAX, YINC

MODULES CALLED: ROUNDSTEP

VARIABLES:	TYPE:	DESCRIPTION:
DEBUG	LOGICAL	INDICATES DEBUG SELECT
DECIBEL	LOGICAL	DECIBEL MAGNITUDE
I	INTEGER*4	COUNTER
XMAX	REAL*4	MAXIMUM X VALUE
XMIN	REAL*4	MINIMUM Y VALUE
YHALF	REAL*4	(YMAX+YMIN)/2
YINC	REAL*4	SCALED GRID INCREMENT
YMAX	REAL*4	MAXIMUM Y VALUE
YMIN	REAL*4	MINIMUM Y VALUE
YSCALE	REAL*4	Y AXIS SCALE
YSTEP	REAL*4	EQUAL TO YINC
YVAL	REAL*4	SCALED GRID LOCATION (Y)

Table A-9. Dictionary for MAG_LABEL

SUBROUTINE: MAGNITUDE_PLOT

FILE NAME: FRPLT.FOR

LANGUAGE: VAX-11 FORTRAN

DESCRIPTION: THIS MODULE USES THE GWCORE TO PLOT THE FREQUENCY RESPONSE MAGNITUDE

CALLING MODULES: FREQOUT

CALLING PROCEDURE:
CALL MAGNITUDE_PLOT (FRDAT, IPOW, NCYC)

PASSED ARGUMENTS: FRDAT, IPOW, NCYC

MODULES CALLED:

CLRSEG	CRRSEG	DELALL	LINA2
LINR2	MAG_LABEL	MOVA2	MOVR2
NEWFRM	PLINA2	SCHJST	SCHPLA
SCHPRE	SCHSIZ	SCHUP2	SFONT
SVPRT2	TEXT		

VARIABLES:	TYPE:	DESCRIPTION:
ALLFLAG	LOGICAL	INDICATES DISPLAY ALL
BODFLAG	LOGICAL	INDICATES DISPLAY BODE
CLOSED	LOGICAL	CLOSED LOOP MODE
DEBUG	LOGICAL	INDICATES DEBUG SELECT
DECIBEL	LOGICAL	INDICATES DECIBEL MODE
DUMMY	REAL*4	DUMMY INPUT VARIABLE
FGRID	REAL*4	SCALED GRID COORD. (X)
FILE(30000)	INTEGER*4	PSEUDO DISPLAY FILE
FLEND	INTEGER*4	LENGTH OF PDF
FRDAT(902,2)	REAL*4	X, Y FREQUENCY DATA
GRID	LOGICAL	INCLUDE GRID ON PLOTS
HERTZ	LOGICAL	HERTZ VS. RADIANS
I	INTEGER*4	COUNTER
IFCOUNT	INTEGER*4	LENGTH OF FRDAT
INSTRG	CHAR*16	INPUT STRING
IPOW	INTEGER*4	POWER OF START FREQ.
K	INTEGER*4	COUNTER
KK	INTEGER*4	COUNTER
NCHARS	INTEGER*4	LENGTH OF INSTRG
NCYC	INTEGER*4	NO. OF LOG CYCLES
NGO	INTEGER*4	FORTRAN OUTPUT UNIT
PLMAX	REAL*4	TEMPORARY MAXIMUM VALUE
PLMIN	REAL*4	TEMPORARY MINIMUM VALUE
SCFAC	REAL*4	SCALE FACTOR

Table A-10. Dictionary for MAGNITUDE_PLOT

SUBROUTINE: MAGNITUDE_PLOT (continued)

VARIABLES:	TYPE:	DESCRIPTION:
XFRDAT(902)	REAL*4	SCALED X DATA
XMAX	REAL*4	MAXIMUM X VALUE
XMAXL	REAL*4	LOG (XMAX)
XMIN	REAL*4	MINIMUM X VALUE
XMINL	REAL*4	LOG (XMIN)
YFRDAT(902)	REAL*4	SCALED Y DATA
YINC	REAL*4	GRID INCREMENT
YMAX	REAL*4	MAXIMUM Y VALUE
YMIN	REAL*4	MINIMUM Y VALUE
YVAL	REAL*4	SCALED GRID COORD. (Y)

Table A-10. Dictionary for MAGNITUDE_PLOT (cont'd)

SUBROUTINE: PHASE_LABEL

FILE NAME: PHPLT.FOR

LANGUAGE: VAX-11 FORTRAN

DESCRIPTION: DETERMINES THE LABELING INTERVAL FOR THE PHASE SHIFT PLOT. THE LABEL VALUES ARE WRITTEN TO DISK FILE "PHLABEL.DAT".

CALLING MODULES: PHASE_PLOT

CALLING PROCEDURE:
CALL PHASE_LABEL (XMIN, XMAX, YMIN, YMAX, YINC)

PASSED ARGUMENTS: XMIN, XMAX, YMIN, YMAX
RETURNED ARGUMENTS: YMIN, YMAX, YINC

MODULES CALLED: ROUNDSTEP

VARIABLES:	TYPE	DESCRIPTION:
DEBUG	LOGICAL	INDICATES DEBUG SELECT
DEGREE	LOGICAL	SELECTS DEGREE MODE
I	INTEGER*4	STEP COUNTER
STEPSIZ	CHAR*5	RADIAN STEP SIZE
XMAX	REAL*4	MAXIMUM X VALUE
XMIN	REAL*4	MINIMUM X VALUE
YHALF	REAL*4	(YMAX+YMIN)/2
YINC	REAL*4	GRID INCREMENT (Y AXIS)
YMAX	REAL*4	MAXIMUM Y VALUE
YMIN	REAL*4	MINIMUM Y VALUE
YSCALE	REAL*4	Y AXIS SCALE
YSTEP	REAL*4	SAME AS YSCALE
YVAL	REAL*4	TEMPORARY GRID VALUE

Table A-11. Dictionary for PHASE_LABEL

SUBROUTINE: PHASE_PLOT

FILE NAME: PHPLT.FOR

LANGUAGE: VAX-11 FORTRAN

DESCRIPTION: PLOTS THE PHASE SHIFT GENERATED BY FREQOUT.

CALLING MODULES: FREQOUT

CALLING PROCEDURE:
CALL PHASE_PLOT (PHDAT, IPOW, NCYC)

PASSED ARGUMENTS: PHDAT, IPOW, NCYC

MODULES CALLED:

CLRSEG	CRRSEG	DELALL	LINA2
LINR2	MOVA2	PHASE_LABEL	NEWFRM
PLINA2	SCHJST	SCHPLA	SCHPRE
SCHSIZ	SCHUP2	SFONT	SVPRT2
TEXT			

VARIABLES:	TYPE:	DESCRIPTION:
ALLFLAG	LOGICAL	INDICATES DISPLAY_ALL
BODFLAG	LOGICAL	INDICATESDISPLAY_BODE
CLOSED	LOGICAL	CLOSED LOOP MODE
DEBUG	LOGICAL	INDICATES DEBUG SELECT
DECIBEL	LOGICAL	INDICATES DECIBEL MODE
DEGREE	LOGICAL	INDICATES DEGREE MODE
DUMMY	REAL*4	DUMMY INPUT VARIABLE
FGRID	REAL*4	SCALED GRID LOCATION
FILE(30000)	INTEGER*4	PSEUDO DISPLAY FILE
FLEND	INTEGER*4	LENGTH OF PDF
GRID	LOGICAL	INDICATES GRID DESIRED
HERTZ	LOGICAL	HERTZ VS RADIANS
I	INTEGER*4	COUNTER
IFCOUNT	INTEGER*4	LENGTH OF PHDAT
INSTRG	CHAR*16	INPUT STRING
IPOW	INTEGER*4	POWER OF STARTING FREQ.
K	INTEGER*4	COUNTER
KK	INTEGER*4	COUNTER
NCHARS	INTEGER*4	LENGTH OF INSTRG
NCYC	INTEGER*4	NO. OF LOG CYCLES
NGO	INTEGER*4	FORTRAN OUTPUT UNIT
PHDAT(902,2)	REAL*4	X, Y SET OF DATA POINTS
PLMAX	REAL*4	TEMPORARY MAXIMUM VALUE
PLMIN	REAL*4	TEMPORARY MINIMUM VALUE
SCFAC	REAL*4	SCALE FACTOR

Table A-12. Dictionary for PHASE_PLOT

SUBROUTINE: PHASE_PLOT (continued)

VARIABLES:	TYPE:	DESCRIPTION:
XMAX	REAL*4	MAXIMUM X VALUE
XMAXL	REAL*4	LOG OF XMAX
XMIN	REAL*4	MINIMUM X VALUE
XMINL	REAL*4	LOG OF XMIN
XPHDAT(902)	REAL*4	SCALED X AXIS DATA
YINC	REAL*4	GRID INCREMENT (Y AXIS)
YMAX	REAL*4	MAXIMUM Y VALUE
YMIN	REAL*4	MINIMUM Y VALUE
YPHDAT(902)	REAL*4	SCALED Y AXIS DATA
YVAL	REAL*4	TEMPORARY GRID VALUE

Table A-12. Dictionary for PHASE_PLOT (cont'd)

SUBROUTINE: ROUNDSTEP

FILE NAME: ROUNDSTEP.FOR

LANGUAGE: VAX-11 FORTRAN

DESCRIPTION: PERFORMS ROUNDING ON THE NUMBER TO BE USED FOR THE GRID SCALE.

CALLING MODULES: LOCUS_PLOT, TIME_LABEL, PHASE_LABEL, MAG_LABEL

CALLING PROCEDURE:
CALL ROUNDSTEP (STEPVAL)

PASSED ARGUMENTS: STEPVAL
RETURNED ARGUMENT: STEPVAL

VARIABLES:	TYPE:	DESCRIPTION:
DEBUG	LOGICAL	INDICATES DEBUG SELECT
I	INTEGER*4	COUNTER
STEPVAL	REAL*4	VALUE/ROUNDED VALUE
YSTEP	REAL*4	ABS(STEPVAL)

Table A-13. Dictionary for ROUNDSTEP

SUBROUTINE: SLIDE

FILE NAME: SLIDE.FOR

LANGUAGE: VAX-11 FORTRAN

DESCRIPTION: DRAWS THE TITLE SLIDE FOR ICECAP-II.

CALLING MODULES: ICER

CALLING PROCEDURE: CALL SLIDE

MODULES CALLED:

CLTSEG	CRTSEG	LINA2	MOVA2
MPICC	SCHJST	SCHPRE	SCHSIZ
SCHSPA	SFONT	SVPRT2	TEXT

VARIABLES:	TYPE:	DESCRIPTION:
DUMMY	REAL*4	DUMMY INPUT VARIABLE

Table A-14. Dictionary for SLIDE

SUBROUTINE: SQUARE

FILE NAME: SQUARE.FOR

LANGUAGE: VAX-11 FORTRAN

DESCRIPTION: THIS MODULE DRAWS A RECTANGLE FOR MAKING BLOCK DIAGRAMS.

CALLING MODULES: FORMER

CALLING PROCEDURE: CALL SQUARE

MODULES CALLED: LINR2 MOVR2

VARIABLES:	TYPE:	DESCRIPTION:
DEBUG	LOGICAL	INDICATES DEBUG SELECT

Table A-15. Dictionary for SQUARE

SUBROUTINE: TIME_LABEL

FILE NAME: TIMPLT.FOR

LANGUAGE: VAX-11 FORTRAN

DESCRIPTION: DETERMINES THE APPROPRIATE GRID LINE SPACING FOR TIME RESPONSE PLOT.

CALLING MODULES: TIME_PLOT

CALLING PROCEDURE:
CALL TIME_LABEL (XMIN, XMAX, YMIN, YMAX, YINC)

PASSED ARGUMENTS: XMIN, XMAX, YMIN, YMAX
RETURNED ARGUMENTS: YMIN, YMAX, YINC

MODULES CALLED: ROUNDSTEP

VARIABLES:	TYPE:	DESCRIPTION:
DEBUG	LOGICAL	INDICATES DEBUG SELECT
GAP	REAL*4	ABS(YMIN)
I	INTEGER*4	COUNTER
INPUTR	INTEGER*4	FORCING FUNCTION
RHIGHT	REAL*4	STRENGTH OF INPUTR
XMAX	REAL*4	MAXIMUM X VALUE
XMIN	REAL*4	MINIMUMX VALUE
XSCALE	REAL*4	X AXIS SCALE
YHALF	REAL*4	(YMAX+YMIN)/2
YINC	REAL*4	GRID SIZE (SCALED)
YMAX	REAL*4	MAXIMUM Y VALUE
YMIN	REAL*4	MINIMUM Y VALUE
YSCALE	REAL*4	Y AXIS SCALE
YSTEP	REAL*4	GRID STEP SIZE
YVAL	REAL*4	SCALED GRID VALUE

Table A-16. Dictionary for TIME_LABEL

SUBROUTINE: TIME_PLOT

FILE NAME: TIMPLT.FOR

LANGUAGE: VAX-11 FORTRAN

DESCRIPTION: PLOTS THE CONTINUOUS TIME RESPONSE USING DATA CALCULATED BY PARTL.

CALLING MODULES: PARTL

CALLING PROCEDURE:
CALL TIME_PLOT (XMIN, XMAX, PLDAT, ICOUNT)

PASSED ARGUMENTS: XMIN, XMAX, PLDAT, ICOUNT

MODULES CALLED:

CLRSEG	CRRSEG	DELALL	LINA2
LINR2	MOVA2	MOVR2	NEWFRM
PLINA2	SCHJST	SCHPLA	SCHPRE
SCHSIZ	SCHUP2	SFONT	SVPRT2
TEXT	TIME_LABEL		

VARIABLES:	TYPE:	DESCRIPTION:
ALLFLAG	LOGICAL	INDICATES DISPLAY_ALL
CLOSED	LOGICAL	CLOSED LOOP MODE
DEBUG	LOGICAL	INDICATES DEBUG SELECT
DUMMY	REAL*4	DUMMY INPUT VARIABLE
FILE(30000)	INTEGER*4	PSEUDO DISPLAY FILE
FLEND	INTEGER*4	LENGTH OF PDF
GRID	LOGICAL	DRAW GRID ON PLOTS
I	INTEGER*4	COUNTER
ICOUNT	INTEGER*4	LENGTH OF PLDAT
INPUTR	INTEGER*4	FORCING FUNCTION
INSTRG	CHAR*16	INPUT STRING
KK	INTEGER*4	COUNTER
NCHARS	INTEGER*4	LENGTH OF INSTRG
NGO	INTEGER*4	FORTRAN OUTPUT UNIT
PLDAT(410,2)	REAL*4	ARRAY OF X, Y DATA
PLMAX	REAL*4	TEMPORARY MAXIMUM
XDAT(410)	REAL*4	SCALED X DATA
XMAX	REAL*4	MAXIMUM X VALUE
XMIN	REAL*4	MINIMUM X VALUE
YDAT(410)	REAL*4	SCALED Y DATA
YINC	REAL*4	GRID SPACING
YMAX	REAL*4	MAXIMUM Y VALUE
YMIN	REAL*4	MINIMUM Y VALUE
YVAL	REAL*4	SCALED GRID VALUE

Table A-17. Dictionary for TIME_PLOT

SUBROUTINE: ZOOM_FLAG

FILE NAME: ZOOMFLAG.FOR

LANGUAGE: VAX-11 FORTRAN

DESCRIPTION: SETS THE LOGICAL VARIABLE ZOOM TO TRUE FOR CONTROL OF VIEWPORT IN LOCUS-PLOT.

CALLING MODULES: LOCUS_ZOOM

CALLING PROCEDURE: CALL ZOOM_FLAG

VARIABLES:	TYPE:	DESCRIPTION:
DEBUG	LOGICAL	INDICATES DEBUG SELECTED
ZOOM	LOGICAL	INDICATES ZOOM OPERATION

Table A-18. Dictionary for ZOOM_FLAG

FILE INFORMATION

All of the files necessary to create and execute ICECAP-II can be found in the Digital Engineering Laboratory on the RK07 disk pack labeled AFITUSER and dated 27 Oct 1983. In the linking process an option file is used which lists the majority of files containing the necessary object code. This option file is called ICER.OPT and can be found in the directory [TRAVIS.MODULES]. A listing of all the files in ICER.OPT is shown in Table A-19. It is also necessary to link with several object libraries such as the GWCORE and the IMSL to produce the complete executable version of ICECAP-II. The following sample link command produces an executable image called ICEII.EXE;

```
LINK/EXECUTABLE=ICEII -
[TRAVIS.MODULES]ICER/OPTION,-
[GEMBAR.MODULES]PLOT10/LIBRARY,-
[GEMBAR.MODULES]PLTUSL/LIBRARY,-
[GEMBAR.MODULES]IMSLD/LIBRARY,-
[GATEWOOD.CORE]GWCORE/LIBRARY
```

It is noted that, although they are not used, the PLOT10 and PLTUSL libraries are linked into the image to avoid unresolved references from portions of VAX_TOTAL which have not yet been modified.

Before ICEII.EXE can be run it is necessary to make several logical assignments for input, output, and file

ADAPT	ADD	ADVANZ	ALL
ALPHA	ANG1	ARROW	AW
AZCALC	AZCOMP	BANG	BIFORM
BILIN	BITERM	BLEND	BLOCKER
BODE	BOX	BOXER	BREAK
CADJB	CALABB	CALCK	CALCTR
CALKEY	CALNUM	CALVAR	CANCEL
CANROOT	CDEXP	CHALK	CIRCLE
CLOSMS	CNTER	COEFF	COMABB
COMCOM	COMEOL	COMNUM	COMOP
COMPOLY	COMVAR	CONVERT	CONVRT
COPYIER	CORINI	CPLXV	CXPAND
DASHER	DATFILL	DBLMULT	DECODER
DEFU	DELETE	DERIV3	DET
DIGITR	DIRIV	DIVI	DNTER
DOLOOP	DS	ECHOS	EVALU3
EVALU8	EXPAND	FACT	FACTO
FINDBORD	FORM	FORMER	FORMT
FRACTOR	FREPLOT	FREQOUT	FREQR
FRPLT	FT	FTOR	FTT
FW	GAIN	GANG1	GENMMPY
GETPOL	GONOGO	GOTOER	GROUP
GTFHTF	HELP	ICER	IDENITY
LISTER	LOCPLT	MADD	MAGNIFY
MATECHO	MATIN	MATMIX	MATOPR
MATRAN	MATRIX	MATS	MATZERO
MINV	MISCELL	MIX	MLTPL
MMPY	MODIFY	MOREMAT	MSQINT
MULT	MULTIP	NICHOLS	NODI
NODV	NOTICE	ONEDV	OPENMS
ORDER3	ORDPOLE	PARTFR	PARTL
PCHAR	PEAK	PHOFS	PHPLT
PLOTE	PLOTFIN	PLOTSET	POLAR
POLE	POLECHO	POLY	POLYADD
POLYCO	POLYM	POLYMLT	POLYSUB
PROPGAT	RDRNUM	READER	READMS
READS	RECALL	RES1	RES2
RES3	RIN	RINIT	ROOT
ROOT10	ROOT11	ROOT12	ROOTS
ROTPLOY	ROUNDSTEP	RSCHAR	RTECHO
SCALER	SEEK	SHRINK	SIMPLE
SIMULAT	SLIDE	SPAXIS	SPECS
SQUARE	STORE	SWAP	SWAPER
SZWROOT	TEKFREQ	TERM	TEST
TFECHO	TIMER	TIMPLT	TITLES
TOTICE	TOTINI	TOTNUM	TRANFER
TRANPOS	TTYPLOT	TWODV	TXCONV
UCIRC	UNPARTL	UPDA	UPDATE
UVECTOR	WMULT1	WMULT2	WMULT3
WPLN	WPOLE	WRITMS	WZBILIN
XFER	XMAT	XVECTOR	YVECTOR
ZEROIN	ZFORMS	ZMULT1	ZOOMDATA
ZOOMFLAG			

Table A-19. Files Listed in ICER.OPT

storage. These assignments have been placed in command file ICEII.COM which can be invoked from the keyboard by typing "@ICEII". The commands included in this file (which also runs the executable image) are;

```
$IF "''F$LOGICAL("PAS$INPUT)'" .NES ""THEN DEASSIGN -
PAS$INPUT
$IF  "''F$LOGICAL("PAS$OUTPUT")'" .NES.  ""  THEN DEASSIGN -
PAS$OUTPUT
$ASSIGN SYS$INPUT PAS$INPUT
$ASSIGN SYS$OUTPUT PAS$OUTPUT
$IF "''F$LOGICAL("FOR006")'" .NES. "" THEN DEASSIGN -
FOR006
$IF "''F$LOGICAL("FOR007")'" .NES. "" THEN DEASSIGN -
FOR007
$ASSIGN ANSWER.DAT FOR006
$ASSIGN SYS$OUTPUT FOR007
$ASSIGN MEMORY.DAT FOR009
$SET DEF DMA1:[TRAVIS.MODULES]
$ASSIGN/USER_MODE SYS$COMMAND: SYS$INPUT:
$RUN ICEII
```

File ANSWER.DAT is used for storing textual output from ICECAP-II while MEMORY.DAT is used to store all problem variables for another session. When a graphical display is to be saved, the GWCORE'S pseudo display file is written to [TRAVIS.PIKFILE]DRAWING.DAT which can then be used with the Metafile Translator or the graphics driver for the line printer. File FOR008.DAT must also exist in the directory in which ICEII.EXE is stored. This file is used for scratch storage during execution.

SUMMARY

This appendix has presented important design documentation for ICECAP-II. The data flow diagrams, structure charts, and data dictionaries document this effort and provide a sound starting point for future efforts. Essential file information has been included and should provide future programmers with the information necessary to update and run more sophisticated versions of ICECAP-II.

APPENDIX B

PRELIMINARY USER'S MANUAL FOR ICECAP-II

Presented to the Faculty of the School of Engineering

of the Air Force Institute of Technology

Air University

in Partial Fulfillment of the

Requirements for the Degree of

Master of Science

by

Mark A. Travis

CAPT USAF

Graduate Electrical Engineering

December 1983

APPENDIX B

TABLE OF CONTENTS

APPENDIX B

PRELIMINARY USER'S MANUAL FOR ICECAP-II

INTRODUCTION

This manual provides the basic instructions necessary for using ICECAP-II. Each of the variables and commands available in this particular implementation are discussed. Finally a sample problem is presented to help new users of the system. This manual describes ICECAP-II as it exists at the end of this effort so various constraints and limitations will be noted.

ICECAP-II'S VARIABLES

ICECAP-II has basically twelve variables with which the user must be concerned: 7 matrices, 4 transfer functions, and 1 scalar. Although no matrix operations are currently available, the seven matrices can be used to store data that will be used in defining other variables. This might include polynomial coefficients or scalar values. The seven matrices have a maximum size of 10x10 and are named:

AMAT	BMAT	CMAT	DMAT
KMAT	FMAT	GMAT	

ICECAP-II has available four transfer functions which

allow a variety of block diagram configurations. The name and purpose of each transfer function is:

GTF = forward transfer function
HTF = feedback transfer function
OLTF = open loop transfer function
CLTF = closed loop transfer function

The only scalar currently used in ICECAP-II is called GAIN. This variable makes it very simple to adjust the forward gain of the control system and observe the effect on the system time and frequency responses.

ICECAP-II'S COMMANDS

Commands are issued to ICECAP-II via the computer terminal's keyboard. Each command consists of at least a verb and most have several modifiers. Commands can be entered any time the "ICECAP>" prompt appears on the display. The entry of these commands can be aborted at any time by entering the dollar sign and a carriage return. In case the complete command cannot be remembered, simply enter the first word of the command. ICECAP-II will respond with all of the valid options. This can be repeated several times if necessary until the command is complete. In the following paragraphs, each command will be listed with a brief description.

The ICECAP-II command must begin with one of the

following verbs:

COPY	FORM	PLOT	RECOVER	TURN
DEFINE	HELP	PRINT	STOP	UPDATE
DISPLAY				

The COPY command is used to copy information from one variable into another. The form of the command is:

COPY (source) (destination)

The valid source and destination variables are:

AMAT	CMAT	PMAT	GTF	KMAT
BMAT	DMAT	GMAT	HTF	OLTF
CLTF				

Transfer functions can only be copied to other transfer functions and matrices can only be copied to other matrices.

The DEFINE command is used to assign values to the twelve variables and to select a forcing function to be used for the time response. The command formats are:

DEFINE (variable)

DEFINE INPUT - to select a forcing function.

The DISPLAY and PRINT commands are used to view any of the variables and the system responses. When the DISPLAY command is issued output is sent only to the user's terminal. The PRINT command causes the output to be written to a file which can be printed once the session is over. Hard copy outputs are discussed later in this manual. The valid commands are listed below (PRINT can be substituted

for DISPLAY in each case):

DISPLAY (variable) - displays the current value of the specified variable.

DISPLAY RESPONSE - displays the continuous time response. The user is prompted for an initial and final time. The horizontal time axis is always divided into 10 divisions.

DISPLAY LOCUS AUTOSCALE - displays the root locus with an automatic scale. The scale is such that all roots are shown.

DISPLAY LOCUS ZOOM - allows the user to zoom in on any particular part of a root locus by identifying the center point and distance to the right hand border. The user will be prompted (see Figure B-3) before the zoom is performed (Figure B-4).

DISPLAY LOCUS SHRINK - shrinks the most recently drawn locus by doubling the vertical and horizontal range of the display.

DISPLAY LOCUS MAGNIFY - magnifies the most recently drawn locus by halving the vertical and horizontal range.

DISPLAY SPECS - displays the figures of merit for the control system (rise time, duplication time, peak time, etc.)

DISPLAY MAGNITUDE - displays the frequency response magnitude. The user is prompted for the power of the starting frequency and number of logarithmic cycles.

DISPLAY PHASE - displays the frequency response phase.

DISPLAY ALL - displays simultaneously the root locus (antoscaled), magnitude, phase, and time response.

DISPLAY BODE - displays simultaneously the magnitude and phase.

The FORM command is used to perform block diagram manipulations. The various options for this command are shown in Figure B-1.

The HELP command is used to obtain information regarding ICECAP-II. The HELP options are:

HELP SYSTEM - to see introductory information about the system.

HELP INITIAL - to see all legal words for starting an ICECAP-II command.

Help for the teaching modules and various commands is not yet available.

The PLOT command is to eventually be used with a CALCOMP plotter. However, a plotter is not available so the implementation and testing of this command is not complete.

The RECOVER command is used to restore the values of the variables from a previous session. There are no modifiers to this command.

The STOP command is the proper way to end an ICECAP-II session. The current values of all the variables are saved before the program actually terminates.

The TURN command controls the status of various "switches" used in ICECAP-II. The possible commands and their effects are:

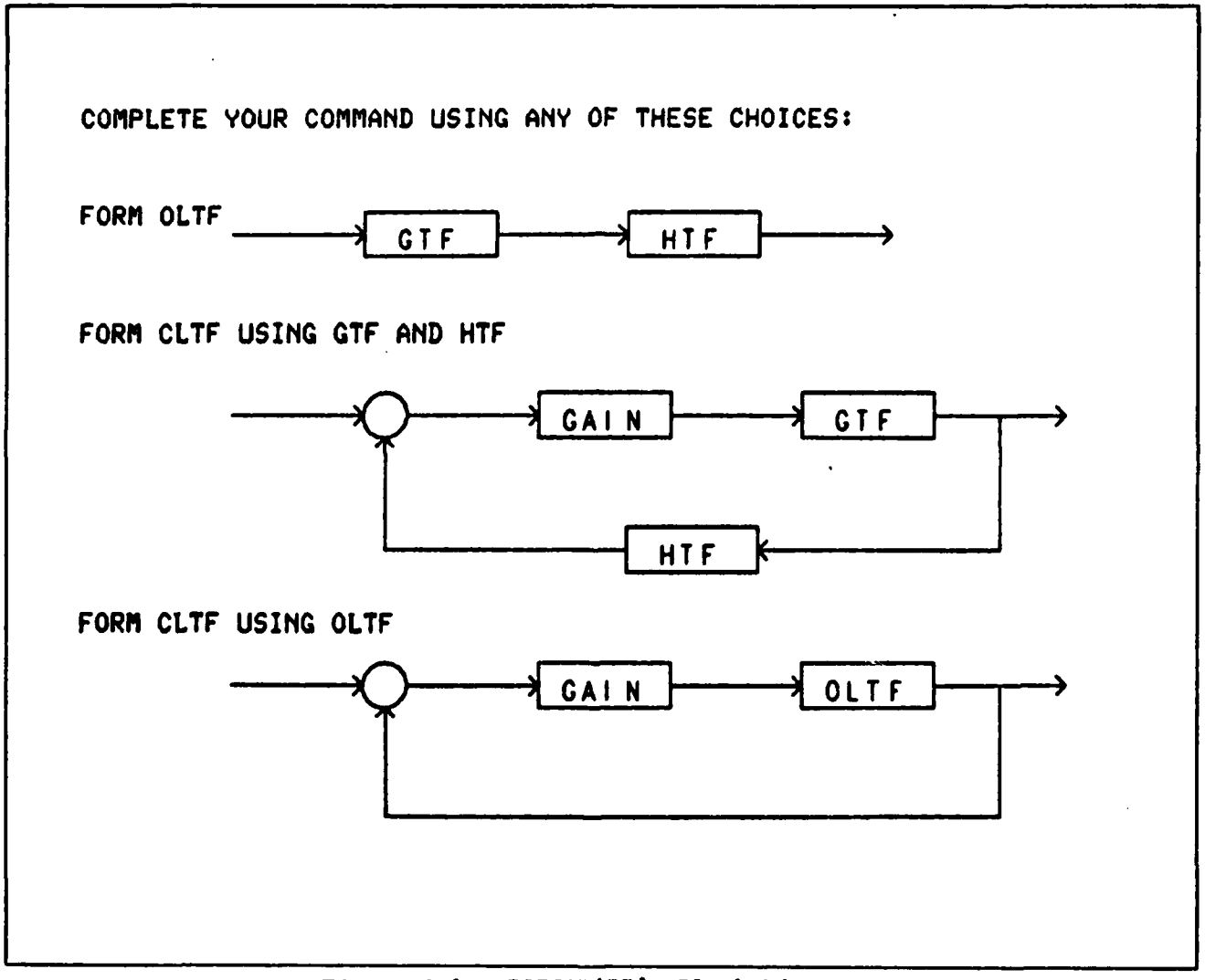


Figure B-1. ICECAP-II's Block Diagrams

TURN ANSWER ON/OFF - when on, causes output to be written to a disk file. This has a similar affect to the PRINT command.

TURN CANCEL ON/OFF - when on, automatically cancels identical poles and zeros.

TURN CAPTION ON/OFF - when on, user may supply a caption for CALCOMP plots.

TURN CLOSED ON/OFF - when on, the CLTF is used for all analysis.

TURN DEBUG ON/OFF - when on, helps users and programmers trace flow of execution.

TURN DECIBELS ON/OFF - when on, frequency response magnitude is calculated in decibels.

TURN ECHO ON/OFF - when on, all input is echoed to the display for confirmation.

TURN GRID ON/OFF - when on, grid lines are drawn on all figures.

TURN HERTZ ON/OFF - when on, frequency response axis is in hertz rather than radians.

TURN MAINMENU ON/OFF - when on, the main ICECAP-II menu is displayed with each prompt.

TURN MULT ON/OFF - when on, multiple CALCOMP plots are drawn on same axes.

TURN PLOT ON/OFF - when on, a CALCOMP plot is automatically generated for each LOCUS command.

TURN SCALE ON/OFF - when on, all CALCOMP plots are autoscaled.

TURN TITLE ON/OFF - when on, a user supplied title is placed on CALCOMP plot.

The UPDATE command is used to save all variables in a disk file at any time desired by the user.

HARD COPY OUTPUT

At any time during the problem solving process a hard copy can be obtained using the Tektronix Hard Copy Unit. Other hard copy output cannot be obtained until the end of a session (after a STOP command has been issued). When any of the variables are PRINTed, they are stored in file ANSWER.DAT. This file can be sent to the line printer by typing;

```
PRINT ANSWER.DAT
```

When graphic displays are PRINTed each is stored in file DRAWING.DAT. These are files of graphic data and must be printed by typing:

```
@DRAW
```

This program will take several minutes to drive the line printer and must be run for each file created.

SAMPLE PROBLEM

To run ICECAP-II, log on to the VAX system and type the following DCL commands:

```
$ SET DEFAULT DMA1:[TRAVIS.MODULES]
$ @ICEII
```

ICECAP-II will respond with a title slide and, after the

user enters a carriage return, the initial menu. The following is a sample of input and output for a third order system. The items which are underlined are user inputs.

```
ICECAP> DEFINE OLTF POLY

  POLYNOMIAL INPUT OF OLTF

  ENTER NUM & DENOM DEGREES (OR SOURCE): > 0,3
  ENTER 1 NUMER COEFF--HI TO LO: >1

  OLTF NUMERATOR (OLNPOLY)          OLTF ZEROS (OLZERO)
(  1.000       )              POLYNOMIAL CONSTANT=  1.000


  ENTER  4 DENOM COEFF--HI TO LO: > .025,0.35,1,0


  OLTF DENOMINATOR (OLDPOLY)        OLTF POLES (OLPOLE)
( 0.2500E-01)S** 3        (  0.0000E+00) + J(0.0000E+00)
( 0.3500    )S** 2        ( -4.000     ) + J(0.0000E+00)
( 1.000     )S** 1        (-10.00      ) + J(0.0000E+00)
( 0.0000E+00)             POLYNOMIAL CONSTANT= 0.2500E-01

   GAIN= 1.0  OLK= GAIN*(OLNK/OLDK)=     40.00000

                     Hit <RETURN> to Continue

ICECAP> DEFINE GAIN

ENTER DESIRED VALUE OF GAIN > 2.94

GAIN=       2.940000057

                     Hit <RETURN> to Continue

ICECAP> FORM CLTF USING OLTF

CLTF = GAIN*OLTF / (1 + GAIN*OLTF)

                     Hit <RETURN> to Continue

ICECAP. DEFINE INPUT

TYPES OF INPUT:
   (1) IMPULSE
```

```
   (2) STEP
   (3) RAMP
   (4) PULSE
   (5) SINE

...SELECT ONE >1

ENTER IMPULSE STRENGTH > 1

                        Hit <RETURN> to Continue

ICECAP> DISPLAY ALL     (see Figure B-2)

FOR CLTF TIME RESPONSE,
ENTER INITIAL TIME, FINAL TIME > 0,10

FOR CLTF FREQUENCY RESPONSE,
ENTER POWER OF STARTING FREQ (-2 FOR .01, ETC) > -1
ENTER NO OF CYCLES TO PLOT (10 MAX) > 3

Valid command words for starting a command are as
follows:

COPY       FORM       PLOT       RECOVER    TURN
DEFINE     HELP       PRINT      STOP       UPDATE
DISPLAY

ICECAP> DISPLAY LOCUS ZOOM    (see Figures B-3 and B-4)

ICECAP> STOP
ALL INFO IN TOTAL HAS BEEN SAVED IN FILE--MEMORY.DAT

$
```

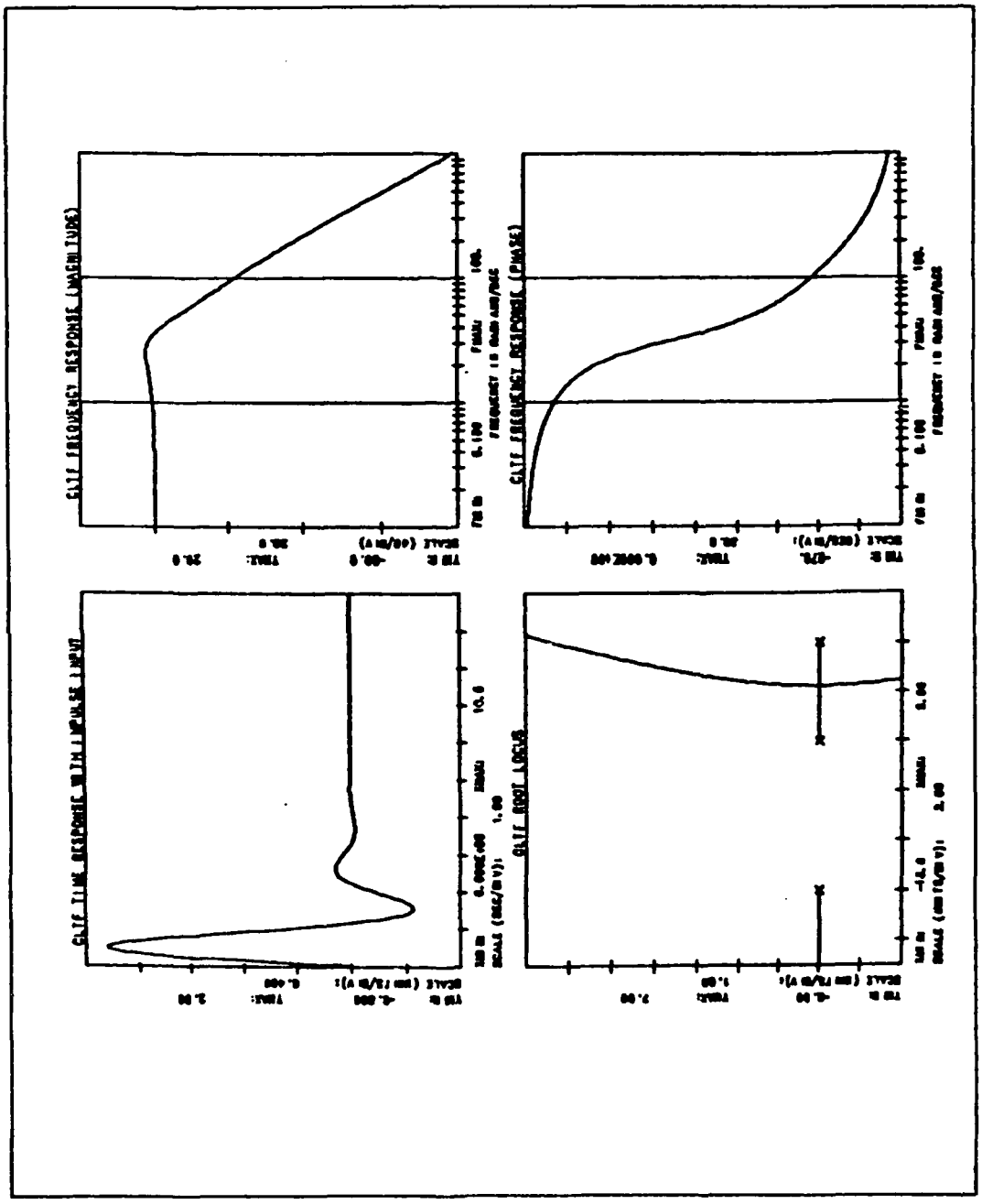


Figure B-2. Response to "DISPLAY ALL" Command

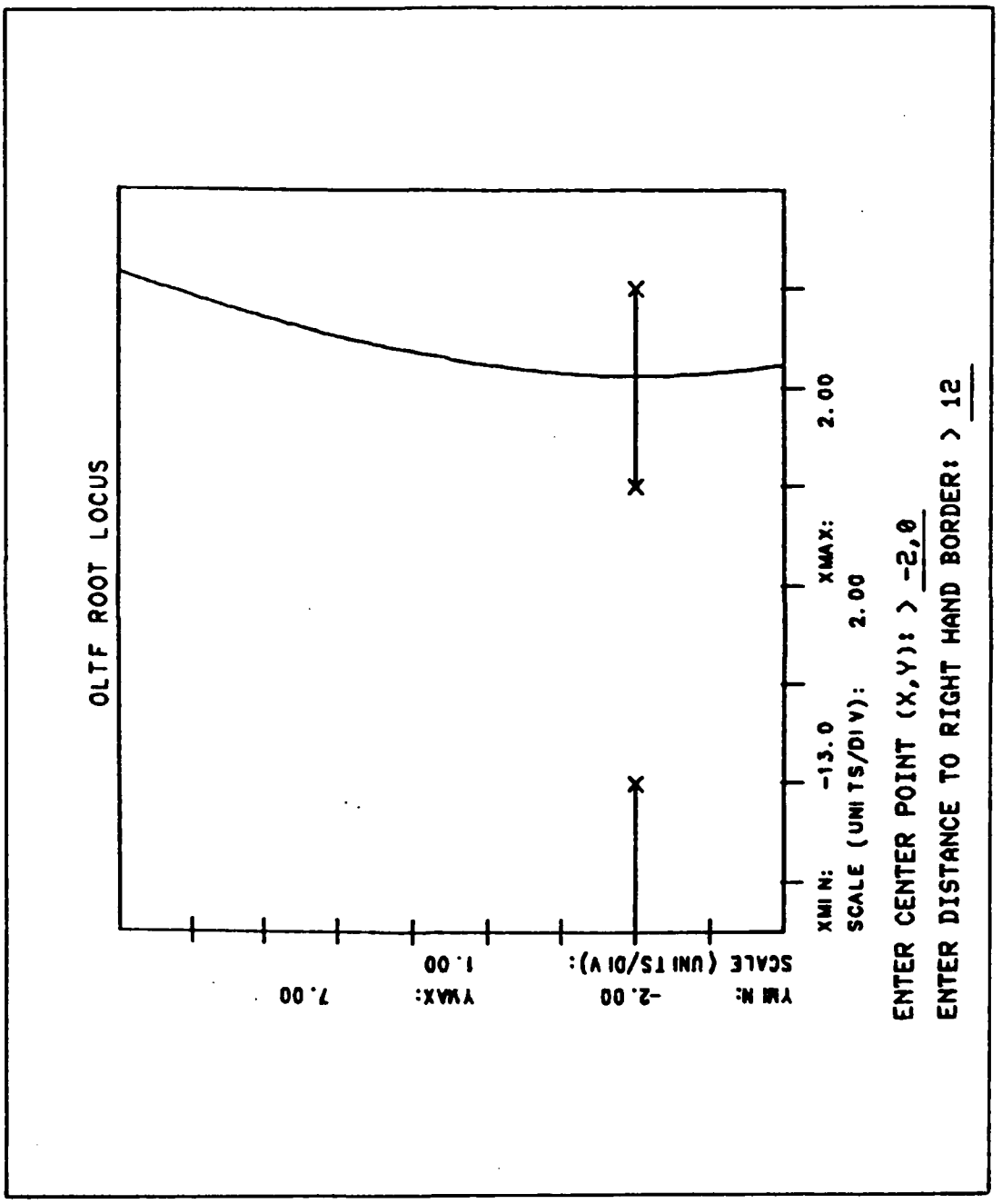


Figure B-3. Initial Response to "DISPLAY LOCUS ZOOM"

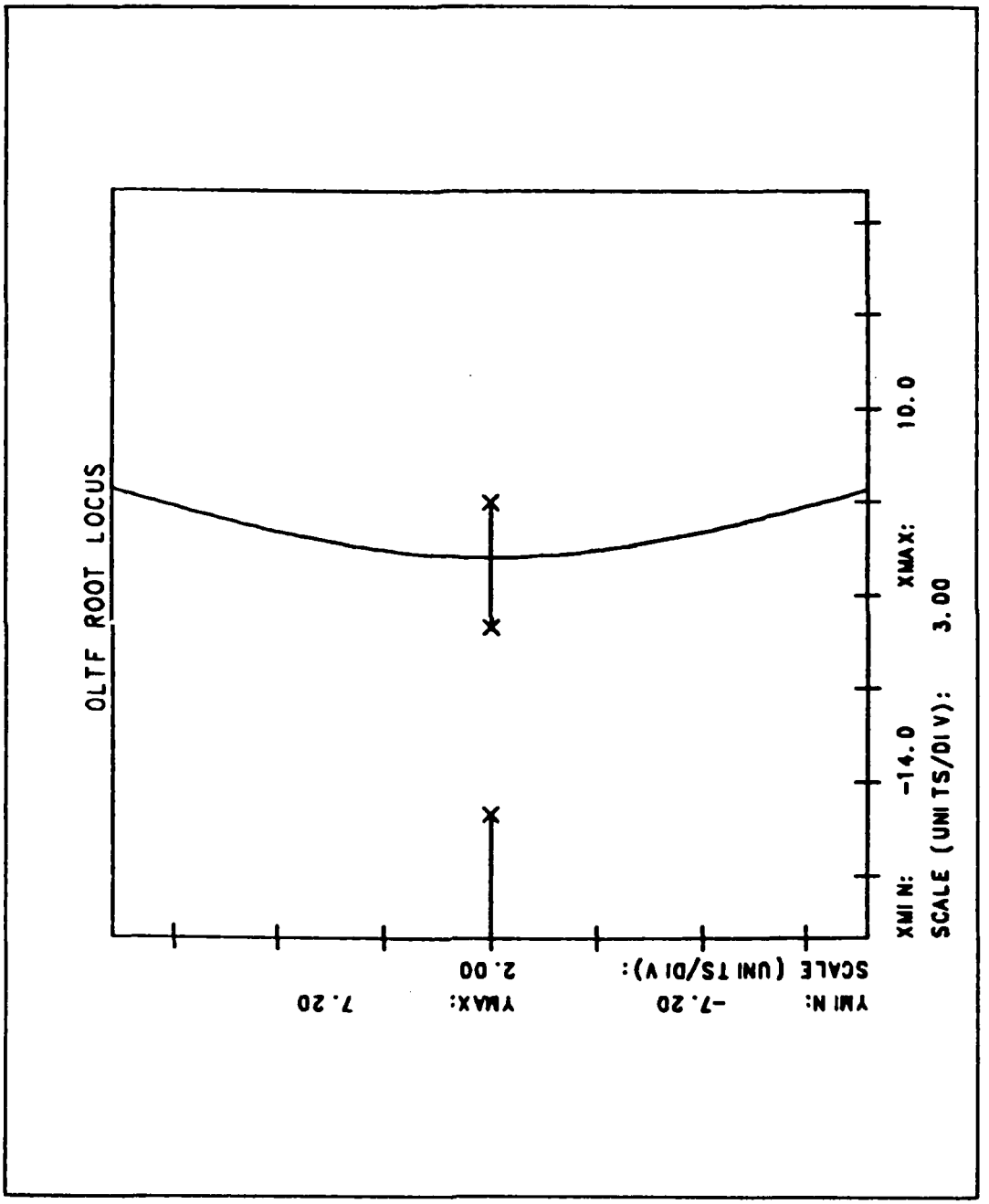


Figure B-4. Final Response to "DISPLAY LOCUS ZOOM"

APPENDIX C

TEST PLAN AND RESULTS FOR ICECAP-II

Presented to the Faculty of the School of Engineering

of the Air Force Institute of Technology

Air University

in Partial Fulfillment of the

Requirements for the Degree of

Master of Science

by

Mark A. Travis

CAPT USAF

Graduate Electrical Engineering

December 1983

APPENDIX C

TABLE OF CONTENTS

APPENDIX C

TESTING & RESULTS

INTRODUCTION

This appendix includes the test plan used for ICECAPII and a variety of sample test results. Any bugs discovered in testing are documented in this appendix along with the proposed fix. Finally, several valid user comments are presented in a discussion of ad-lib testing.

TEST PLAN

No additional equipment is required for this testing as all the necessary hardware is currently available in the Digital Engineering Laboratory. Testing is performed using a Tektronix 4014-1 display and a 4631 hard copy unit. The version of ICECAP being tested is dated 7 Oct 1983. The following steps represent the basic plan for testing ICECAP-II from a functional or black-box point of view.

I. GRAPHIC PERFORMANCE

1. Modify the ICECAP-II source code to write the set of X-Y data points to a disk file before drawing the graph.

2. Run ICECAP-II and turn the grid switch on.

3. Enter a transfer function and obtain a hard copy of each of the four basic graphic displays: time response, magnitude plot, phase plot, and root locus.

4. Exit from ICECAP-II and print the disk files containing the X-Y data.

5. Compare the X-Y data to the graphic hard copy. Verify points that are easy to read on the graph such as maximum values, minimum values, and intersections with grid lines.

6. If the graph accurately represents the data (within 1% or 2% of the data values), this test is finished.

7. If the graph does not accurately represent the data, determine if the fault is a software bug or a limitation of the hardware.

8. Correct any software bugs and return to Step 1

II. BLACK-BOX TESTING

1. Run the ICECAP-II program and execute each valid command.

2. If each valid command does not produce the desired

result, find and correct the software bug and return to Step 1.

3. At various data or command entry points, provide invalid input from each of the following cases.

    a. Enter an incorrect input.

    b. Enter an improper combination of inputs.

    c. Enter insufficient input.

    d. Enter no input at all.

    e. Enter too much.

4. Collect samples for the various cases. Always collect a sample (hard copy) when a bug is discovered.

5. Correct any bugs and return to Step 3.

III. AD-LIB TESTING

There is no plan for this type of test since it is conducted on an informal basis by other individuals.

SAMPLE RESULTS

Sample results for each of the three categories are contained in the following pages.

I. GRAPHIC PERFORMANCE

A sample of each graphic output and corresponding list of data points checked is included in the next eight pages.

II. BLACK-BOX TESTING

Two samples for each of the five input cases are listed below.

a. Incorrect Input

1. ICECAP>DEFINE INPUT

```
TYPES OF INPUT:
     (1) IMPULSE
     (2) STEP
     (3) RAMP
     (4) PULSE
     (5) SINE

...SELECT ONE >6
ERROR--ENTER 1,2,3,4,OR5:

TYPES OF INPUT:
     (1) IMPULSE
     (2) STEP
     (3) RAMP
     (4) PULSE
     (5) SINE

...SELECT ONE >
```

2. ICECAP> DEFINE GAIN

```
ENTER DESIRED VALUE OF GAIN > QMAT(1,1)

ERROR>QMAT
WAITING FOR DATA INPUT AFTER EQUAL SIGN
ERROR>QMAT
WAITING FOR DATA INPUT AFTER EQUAL SIGN
ERROR>QMAT
WAITING FOR DATA INPUT AFTER EQUAL SIGN
```

A bug is found in this sample as the error message is repeated continuously. This is a severe error as the program must be terminated. The input routine must be examined to correct this error.

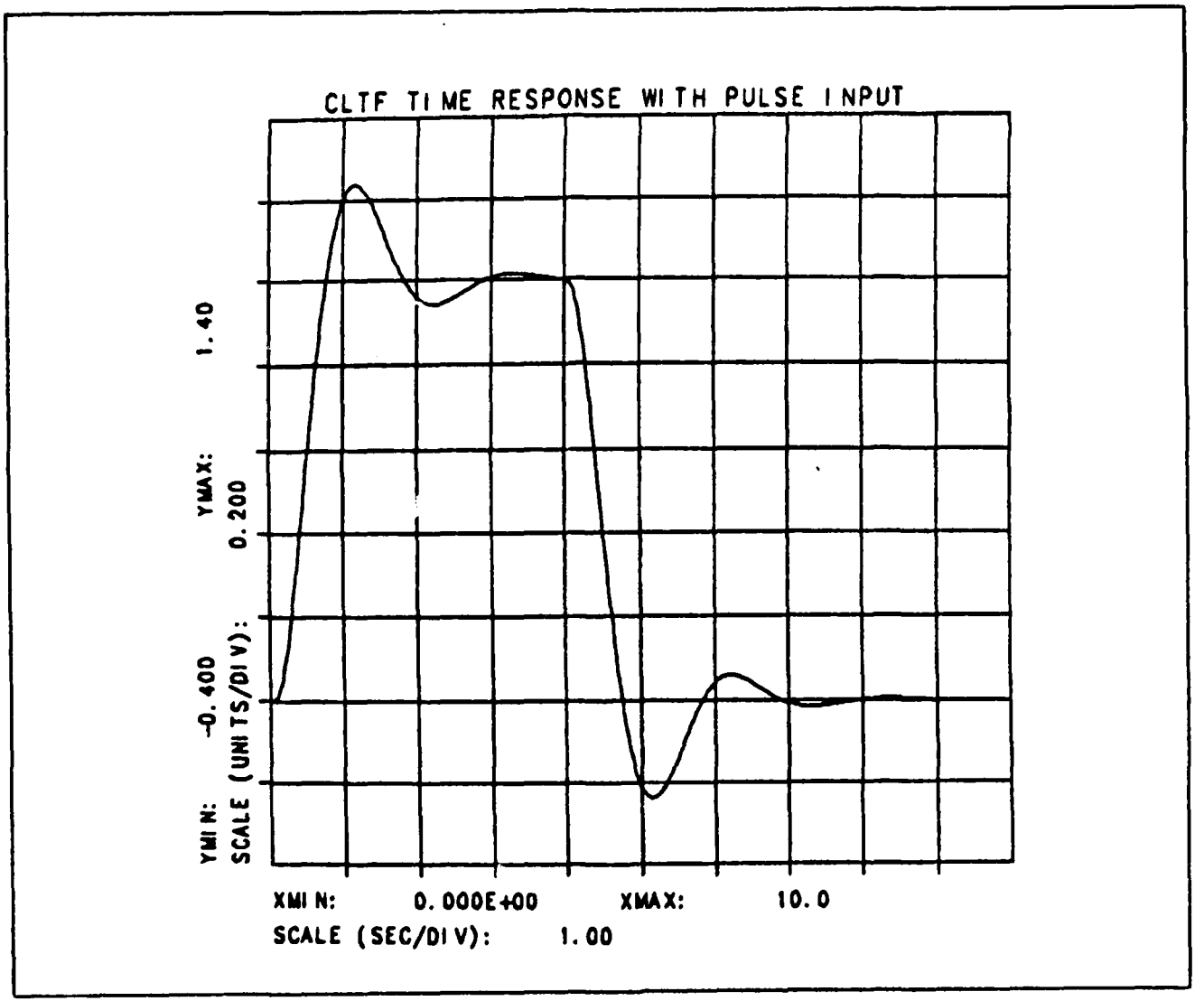


Figure C-1. Sample Time Response

Time (sec)	Output	
0.00000E+00	0.00000E+00	
0.30000E+00	0.20761E+00	
0.52500E+00	0.60078E+00	
0.77500E+00	0.10020E+01	
0.10000E+01	0.11991E+01	
0.11750E+01	0.12373E+01	(max)
0.20000E+01	0.96104E+00	
0.29250E+01	0.10009E+01	
0.30000E+01	0.10060E+01	
0.40000E+01	0.99967E+00	
0.50000E+01	-0.19933E+00	
0.51750E+01	-0.23682E+00	(min)
0.60000E+01	0.39127E-01	
0.70000E+01	-0.60953E-02	
0.80000E+01	0.35256E-03	
0.90000E+01	0.26304E-03	
0.10000E+02	-0.16618E-03	

Table C-1. Time Response Data Points

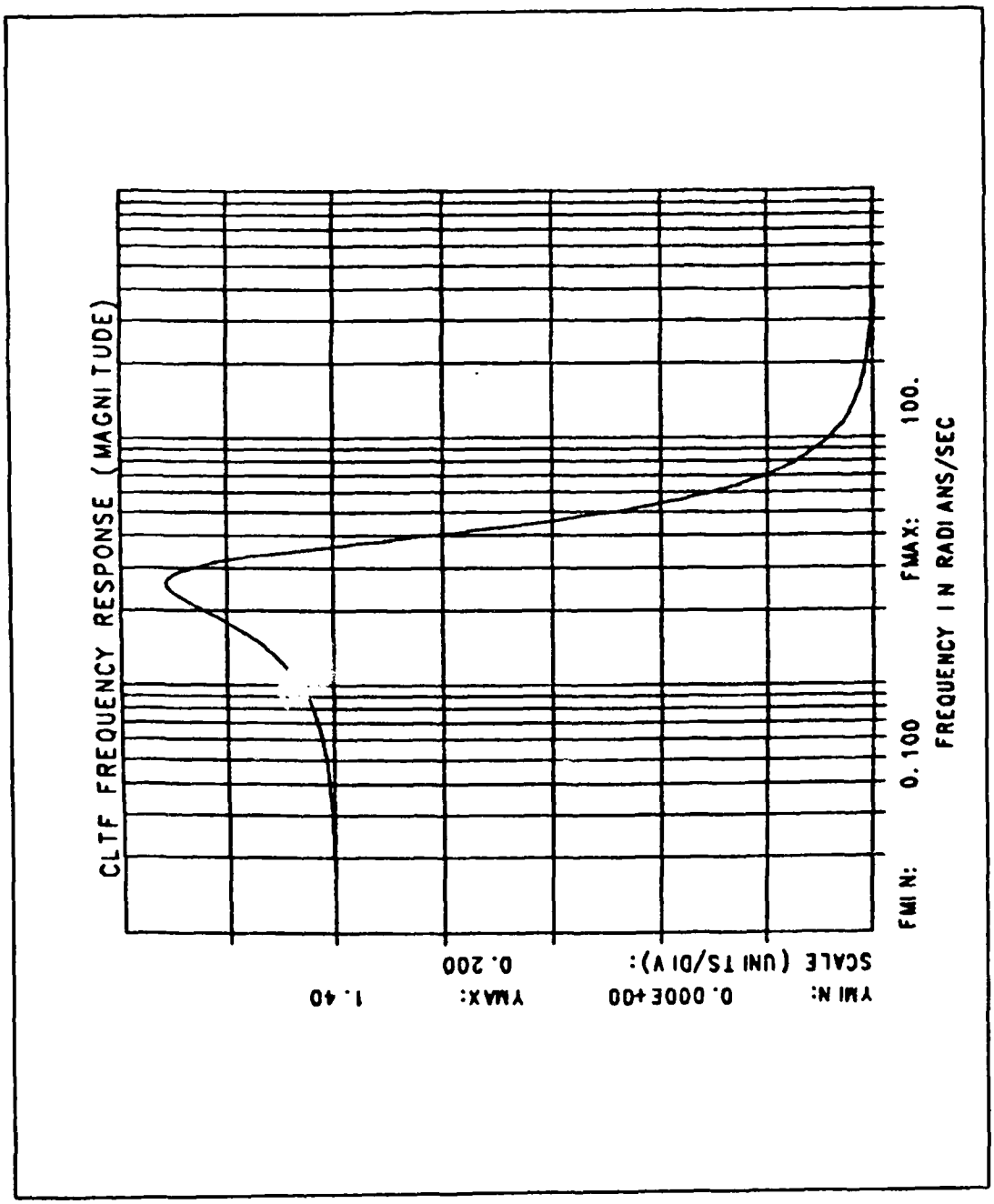
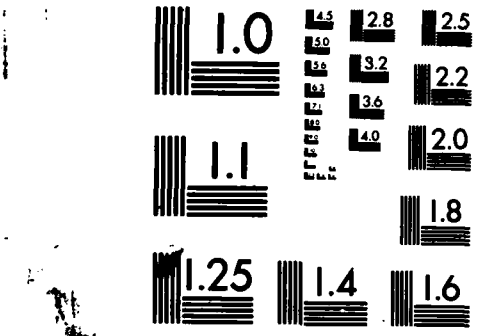


Figure C-2. Sample Magnitude Plot

Freq (rad/sec)	Magnitude	
0.10000000	1.0006125	
0.20000000	1.0024507	
0.30000001	1.0055199	
0.40000001	1.0098280	
0.50000000	1.0153860	
0.60000002	1.0222059	
0.69999999	1.0303013	
0.80000001	1.0396851	
0.90000004	1.0503675	
1.0000000	1.0623540	
1.8000000	1.2004490	
2.0000000	1.2410922	
2.5999999	1.3151147	(max)
3.0000000	1.2593894	
3.2000000	1.1920388	
4.0000000	0.82061464	
5.0000000	0.48156798	
6.0000000	0.30376247	
7.0000000	0.20563732	
8.0000000	0.14668286	
9.0000000	0.10875703	
10.000000	0.83061337E-01	
20.000000	0.12994937E-01	
30.000000	0.41031437E-02	
40.000000	0.17748511E-02	
50.000000	0.91982336E-03	
60.000000	0.53591310E-03	
70.000000	0.33888116E-03	
80.000000	0.22763843E-03	
90.000000	0.16017606E-03	
100.00000	0.11692477E-03	

TABLE C-2. Magnitude Plot Data Points

AD-A138 025 INTERACTIVE COMPUTER GRAPHICS FOR SYSTEM ANALYSIS(U) 3/3
AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOOL
OF ENGINEERING M A TRAVIS DEC 83 AFIT/GE/EE/83D-66
UNCLASSIFIED F/G 5/1 NL



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

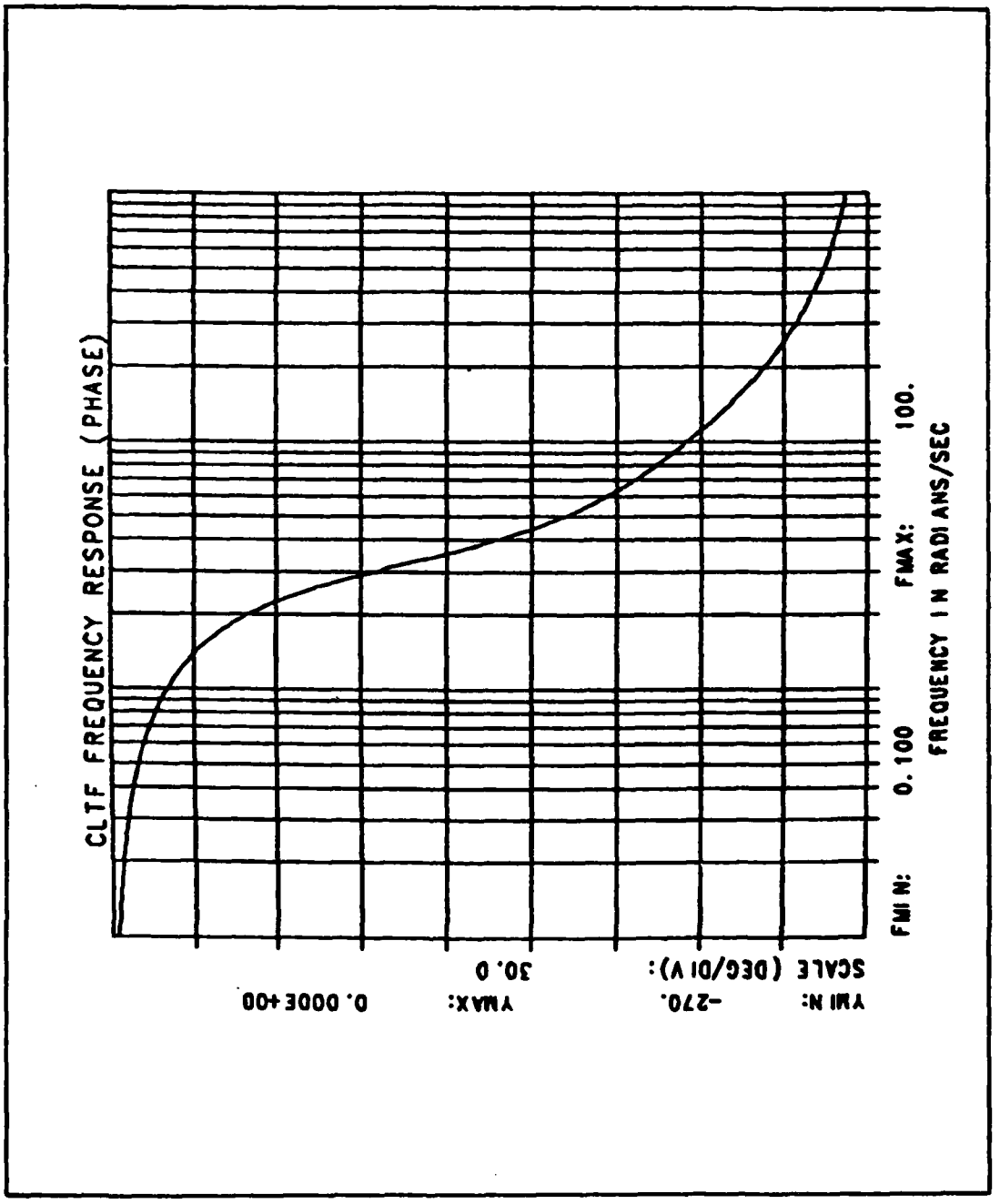


Figure C-3. Sample Phase Plot

Freq (rad/sec)	Phase (degrees)
0.10000000	-1.9499154
0.20000000	-3.9063377
0.30000001	-5.8758421
0.40000001	-7.8651543
0.50000000	-9.8811712
0.60000002	-11.931137
0.69999999	-14.022606
0.80000001	-16.163603
0.90000004	-18.362616
1.0000000	-20.628746
1.4000000	-30.569643
2.0000000	-49.451183
3.0000000	-95.161110
3.5000000	-119.02831
4.0000000	-137.94147
5.0000000	-162.11411
6.0000000	-176.44582
6.4000001	-180.77220
7.0000000	-186.32469
8.0000000	-193.85597
9.0000000	-199.95322
10.000000	-205.07361
11.000000	-209.47565
20.000000	-232.71272
26.000000	-240.52420
30.000000	-244.18166
40.000000	-250.34888
50.000000	-254.16684
60.000000	-256.75348
70.000000	-258.61844
80.000000	-260.02542
100.00000	-262.00543

Table C-3. Phase Plot Data Points

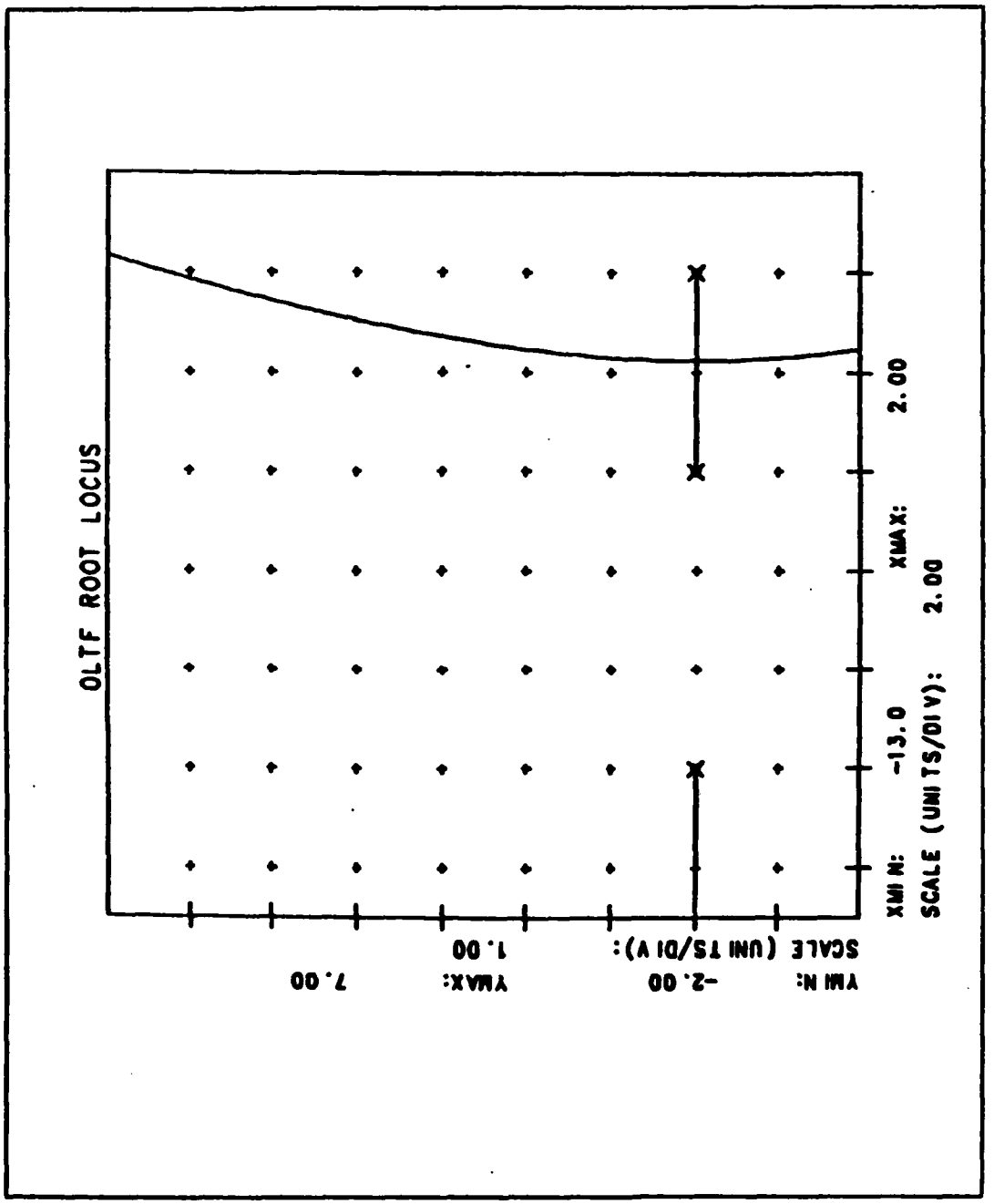


Figure C-4. Sample Root Locus

Real Axis	Imaginary Axis
0.00000000E+00	0.00000000E+00
-1.0000001	0.00000000E+00
-1.7607340	0.00000000E+00
-1.6924904	1.0971954
-1.5218976	2.0821593
-1.0219722	3.8102586
0.00000000E+00	6.3245463
-4.0000000	0.00000000E+00
-1.7607340	0.00000000E+00
-1.7041734	-0.99788016
-1.5218974	-2.0821590
-9.9999990	-2.0821590
-12.000007	0.00000000E+00
-13.000010	0.00000000E+00

Table C-4. Root Locus Data Points

b. Improper Combination of Inputs

3. ICECAP> DISPLAY FORM
DISPLAY FORM is not a valid ICECAP command.

Hit <RETURN> to Continue

4. ICECAP> DISPLAY RESPONSE
ICECAP CONTINUOUS TIME RESPONSE FOR CLTF(S)
WITH STEP INPUT OF STRENGTH = 1.000000

ENTER INITIAL TIME, FINAL TIME > 0,B

ERROR - ILLEGAL INPUT COMMAND
0.0000000E+00

MORE DATA PLEASE

c. Insufficient Input

5. ICECAP> DISPLAY RESPONSE

ICECAP CONTINUOUS TIME RESPONSE FOR CLTF(S)
WITH STEP INPUT OF STRENGTH = 1.000000

ENTER INITIAL TIME, FINAL TIME > 0

MORE DATA PLEASE

6. ICECAP > DEFINE

ENTER choice of Transfer Function: gtf, htf, oltf, cltf
Enter choice of Matrix: AMAT, BMAT, CMAT, DMAT, FMAT, GMAT, KMAT

Enter SETUP in order to set up State Space Model

Enter INPUT to select a forcing function

Enter GAIN to input a desired value

or

Enter $ to abort command.

ICECAP> DEFINE

d. No Input (carriage return only)

7. ICECAP> DEFINE GAIN
ENTER DESIRED VALUE OF GAIN >

```
>
ERROR>GAIN
TOO MANY EQUAL SIGNS
>
ERROR>GAIN
TOO MANY EQUAL SIGNS
>
ERROR>GAIN
TOO MANY EQUAL SIGNS
```

This severe error is related to the other error since it occurred for the same command. The program also had to be terminated to stop the error message.

8.
```
ICECAP> DISPLAY MAGNITUDE

ICECAP CLOSED-LOOP FREQUENCY RESPONSE
ENTER POWER OF STARTING FREQ (-2 FOR .01, ETC) >

MORE DATA PLEASE
```

e. Enter Too Much

9.
```
ICECAP> DISPLAY LOCUS AUTOSCALE ZOOM
```

10.
```
ICECAP CONTINUOUS TIME RESPONSE FOR CLTF(S)
WITH STEP INPUT OF STRENGTH =    1.000000

ENTER INITIAL TIME, FINAL TIME > 0,10,20
```

In each of the above cases the first valid command or data was accepted by the system.

III. AD-LIB TESTING

Although no new bugs were discovered during the ad-lib testing, the following general comments were received from various users:

1. There is no abort mechanism in the data input mode (only the command input mode). This feature is available in TOTAL but not in VAX TOTAL or ICECAP.

2. Some users experienced with using TOTAL prefer to set

their own borders for the root locus rather than use the autoscale or zoom feature.

3. A more suitable grid line needs to be used for the root locus. Since a solid grid line is not practical (as it may cover a trace) a dotted or dashed line should be examined.

4. The original title slide for ICECAP-II is too slow taking nearly 30 seconds to create. This problem has since been corrected by changing to string-precision text. Execution time is now two to three seconds.

SUMMARY

Although ICECAP-II is not completely bug free, the user is provided a great deal of input protection. This testing has demonstrated the usefulness of the five test input cases for finding the most common types of errors. Those errors which cause or lead to program termination are the most serious and deserve the highest priority in debugging.

VITA

Mark Alan Travis was born on New Years Day, 1957 in Morgantown, West Virginia. After graduating from Parkersburg High School, Parkersburg, West Virginia, he attended West Virginia University in Morgantown. He was awarded a Bachelor of Science degree in Electrical Engineering and entered the Air Force as a second lieutenant in May 1979. Following six months of technical training at Keesler Air Force Base, he was assigned to the Telecommunications Directorate of the Electronic Systems Division at Hanscom AFB, Massachusetts. Here, he served as Project Management Officer for the SPEAKEASY program. He enrolled in the Air Force Institute of Technology in June 1982.

Permanent Address: 4 Woodland Drive
Vienna, WV 26105

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

AD-A 138 025

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION: UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT: Approved for public release; distribution unlimited
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S): AFIT/GE/EE/83D-66	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION: School of Engineering; 6b. OFFICE SYMBOL (If applicable): AFIT/ENG	7a. NAME OF MONITORING ORGANIZATION
6c. ADDRESS (City, State and ZIP Code): Air Force Institute of Technology, Wright Patterson AFB, Ohio 45433	7b. ADDRESS (City, State and ZIP Code)
8a. NAME OF FUNDING/SPONSORING ORGANIZATION; 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State and ZIP Code)	10. SOURCE OF FUNDING NOS. — PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT NO.
11. TITLE (Include Security Classification): See Box 19	
12. PERSONAL AUTHOR(S): Mark A. Travis, B.S., Capt, USAF	
13a. TYPE OF REPORT: MS Thesis; 13b. TIME COVERED: FROM _____ TO _____	14. DATE OF REPORT (Yr., Mo., Day): 1983 December; 15. PAGE COUNT: 201
16. SUPPLEMENTARY NOTATION	Approved for public release: IAW AFR 190-17. LYNN E. WOLAVER, Dean for Research and Professional Development, Air Force Institute of Technology (AIC), Wright-Patterson AFB OH 45433, 7 Feb 84
17. COSATI CODES: FIELD 09; GROUP 02; SUB. GR.	18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number): Computer Graphics, Control Systems, Core Standard, System Analysis

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

Title: INTERACTIVE COMPUTER GRAPHICS FOR SYSTEM ANALYSIS

Thesis Chairman: Dr. Gary B. Lamont

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT: UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT. ☐ DTIC USERS ☐	21. ABSTRACT SECURITY CLASSIFICATION: UNCLASSIFIED
22a. NAME OF RESPONSIBLE INDIVIDUAL: Dr. Gary B. Lamont	22b. TELEPHONE NUMBER (Include Area Code): 513-255-3576; 22c. OFFICE SYMBOL: AFIT/ENG

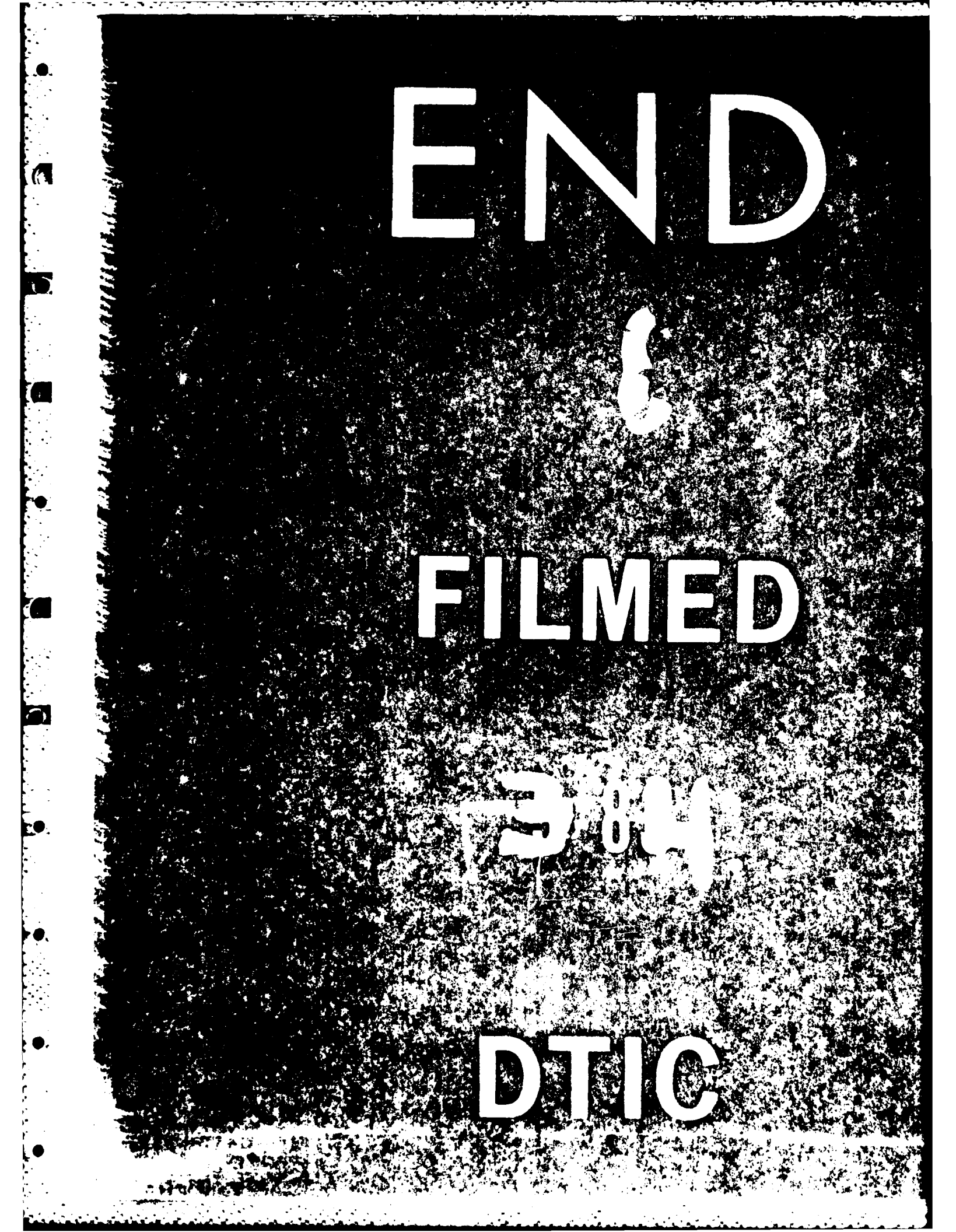
DD FORM 1473, 83 APR — EDITION OF 1 JAN 73 IS OBSOLETE.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

19. ABSTRACT

The possible applications of computer graphics for control system design are considered in this thesis. The functional requirements of such a system are identified and discussed. Other topics such as graphics software, programming languages, user interface, and software design are vital to such an effort and are examined in detail. Based on the desirable features of device independent graphics, an implementation of the ACM/SIGGRAPH Core Standard Proposal is selected. To demonstrate the various applications for system analysis, a group of functions for analysis of continuous systems is implemented and tested. Sample results are provided and recommendations for further work are discussed.


END
FILMED
DTIC